VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

VÝUKOVÁ APLIKACE STRÁNKOVÁNÍ PAMĚTI

EDUCATIONAL APPLICATION ON PAGING SYSTEM

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. PETR NECHVÁTAL**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. ALEŠ SMRČKA, Ph.D.**
SUPERVISOR

BRNO 2017

Abstrakt

Tato práce se zabývá návrhem a implementací výukové aplikace pro stránkování paměti. Úkolem této aplikace je pomoci studentům pochopit a procvičit si některé koncepty ohledně stránkování paměti. Umožní studentům napsat části těchto konceptů a vidět na vizualizaci, jak jejich kód pracuje při simulaci paměťového systému. Aplikace bude implementována jako webová aplikace v HTML, CSS a JavaScriptu. Server se bude starat o překlad uživatelského kódu, bude desktopová aplikace. Tato práce se zaměřuje na popis stránkování paměti a technologie, které byly použity pro tuto práci a návrhem aplikace. Také obsahuje popis implementace a testování této práce.

Abstract

This master's thesis deals with design and implementation of educational application for paging. Goal of the application is to help students understand and practice some concepts from paging. It will allow students to write parts of these concepts and see how their code work on visualization of simulation of memory system. Application will be implemented as a web application in HTML, CSS and JavaScript. Server, which will be taking care of compiling of user code will be a desktop application. This thesis mainly describes paging and technologies which will be used for this thesis and application design. It also describes implementations and testing of this work.

Klíčová slova

React, JavaScript, jednostránková aplikace, stránkování, virtuální paměť, jednotka správy paměti, MMU, websokety, tabulky stránek, výpadek stránky.

Keywords

React, JavaScript, single page application, paging, virtual memory, MMU, websockets, page tables, page fault.

Citace

Nechvátal Petr: Výuková aplikace stránkování paměti, diplomová práce, Brno, FIT VUT v Brně, 2017

Výuková aplikace stránkování paměti

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Aleše Smrčka, Ph.D. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

. . . . . .

Petr Nechvátal

31. července 2017

Poděkování

Rád bych poděkoval svému vedoucímu panu Ing. Aleši Smrčkovi, Ph.D. za detailní popsání zadání a požadavků na aplikaci. Dále bych rád poděkoval svému spolubydlícímu Martinu Veselému za poskytnutí optimálního prostředí pro psaní diplomové práce. Dále bych rád poděkoval Lucce Novotné za korekturu a pomoc s pravidly pravopisu.

Obsah

Kapitola 1

Úvod

Cílem této práce je vytvoření výukové aplikace se zaměřením na stránkování paměti. Jedná se o implementaci webové aplikace, která umožní představení konceptů a algoritmů z oblasti stránkování paměti interaktivní cestou. Aplikace je zaměřena na studenty, kterým poslouží jako nástroj pro vyzkoušení konceptů naučených z výuky. Některým z konceptů, které bude aplikace představovat, je například manipulace s tabulkou stránek, obsluha výpadků stránek, stránkování na žádost, sdílení paměti. Aplikace umožní studentům naimplementovat části těchto konceptů a zobrazí jim, jak stránkování funguje.

Přínosem této práce bude zajímavý prvek ve výuce studentů ve stylu školy hrou. Tím, že budou moci něco vytvořit a okamžitě vidět, co se děje na pozadí, doufejme, povede jak k prohloubení jejich znalostí, tak i zájmu o danou tématiku.

Tato práce seznámí čtenáře s vyvíjenou aplikací. V následující kapitole je nastíněná teorie použitá pro tuto práci. Jedná se zejména o popis paměťového systému a stránkování, je tam popsána většina základních konceptů i několik algoritmů. Dostatečně obsáhlé, aby čtenář pochopil, o čem stránkování je, jak funguje, a proč se využívá. Dále jsou v této kapitole také popsány technologie, které budou použité pro vyvíjení aplikace. Jedná se především o webové technologie, kde jsou popsány nástroje využité pro tvorbu aplikace, definováno několik pojmů, jako je například jednostránková aplikace, a stručně popsány základní technologie pro tvorbu webových stránek jako jsou HTML, CSS či JavaScript, SVG. Také se zde rozeberou linuxové kontejnery.

Následuje kapitola s návrhem aplikace. V této kapitole jsou popsány požadavky na aplikaci a shrnuto, jakou funkcionalitu by měla nabízet. Dále je zde vysvětlena struktura systému, kde je popsáno, jak aplikace bude rozdělená do částí a popsané jednotlivé části. Kapitola také obsahuje několik diagramů reprezentující návrh systému.

Pak následuje kapitola se samotnou implementací. Zde je popsáno, jak byl systém vytvořen. Jsou tam popsány využité technologie, vysvětlena rozhodnutí udělaná během implementace. U každé části systému je pak detailně popsáno, co všechno ta určitá část umí. Co dělá a jak byla implementována. Tato kapitola obsahuje mimo jiné i popis, jak tento systém zprovoznit a jaké má požadavky pro chod. Je zde také popsáno, jak byla vytvořena testovací sada a jak se aplikace testovalo. Dotknuto je tu i téma bezpečnosti aplikace.

Poslední kapitolu tvoří závěr, který shrnuje, čeho bylo v této práci dosaženo. V jakém stavu je daná práce a navrhuje i nápady na rozšíření a případné další ubírání této práce.

Kapitola 2

Teoretický úvod do správy paměti

Tato kapitola obsahuje teorii této diplomové práce. Probírají se tu dvě hlavní témata, a to správa paměti a webové technologie použité pro tuto práci.

V první části jsou tu popsané koncepty paměťového systému, je tu vysvětleno, proč se používají, jaké mají výhody a jak se k nim došlo. Nejdůležitější část pro tuto diplomovou práci je především stránkování a popis virtuální paměti, ale jsou tu popsané i jiné základnější koncepty správy paměti, které slouží především pro ilustraci, proč je stránkování a virtuální paměť za potřebí a jaké jsou jejich výhody. Podrobně jsou tu také popsány tabulky stránek, složky stránek a jednotka správy paměti (Memory Management Unit). Další koncepty související se stránkováním, které jsou zde popsané, jsou TLB (Translation Lookaside Buffer) a algoritmy pro nahrazování stránek.

V druhé části se práce věnuje webovým technologiím a nástrojům použitých pro tvorbu webových aplikací, které jsou využity pro tuto práci. Od základních technologií jako jsou HTML, CSS, AJAX a JavaScript se také podíváme na koncept vývoje jednostránkových aplikací a technologie pro přenos dat mezi server a klientem websockety a JSON. Také jsou tu popsané některé nástroje, které byly použité jako Bootstrap pro design stránek a SVG pro vizualizaci stránkování.

Na závěr bude ještě zmínka o linuxových kontejnerech.

2.1 Správa paměti

Správce paměti je část operačního systému, která se stará o hierarchii paměti. Jeho práce je sledovat, které části paměti jsou používané a které nepoužívané, alokovat paměť procesům, když ji potřebují a uvolnit paměť, když ji přestanou potřebovat. Také se stará o přenos dat mezi pamětí a diskem v případech, kdy je hlavní paměť příliš malá na to, aby udržela data všech procesů.

2.1.1 Jednoprogramový systém

Pro jednoduchý systém, který umožňuje běh pouze jednomu programu, lze použít velice jednoduchý systém správy paměti. Paměť je sdílena mezi programem a operačním systémem. Systém nahraje program příkazu zadaného uživatelem a vykoná jej. Systém po dokončení

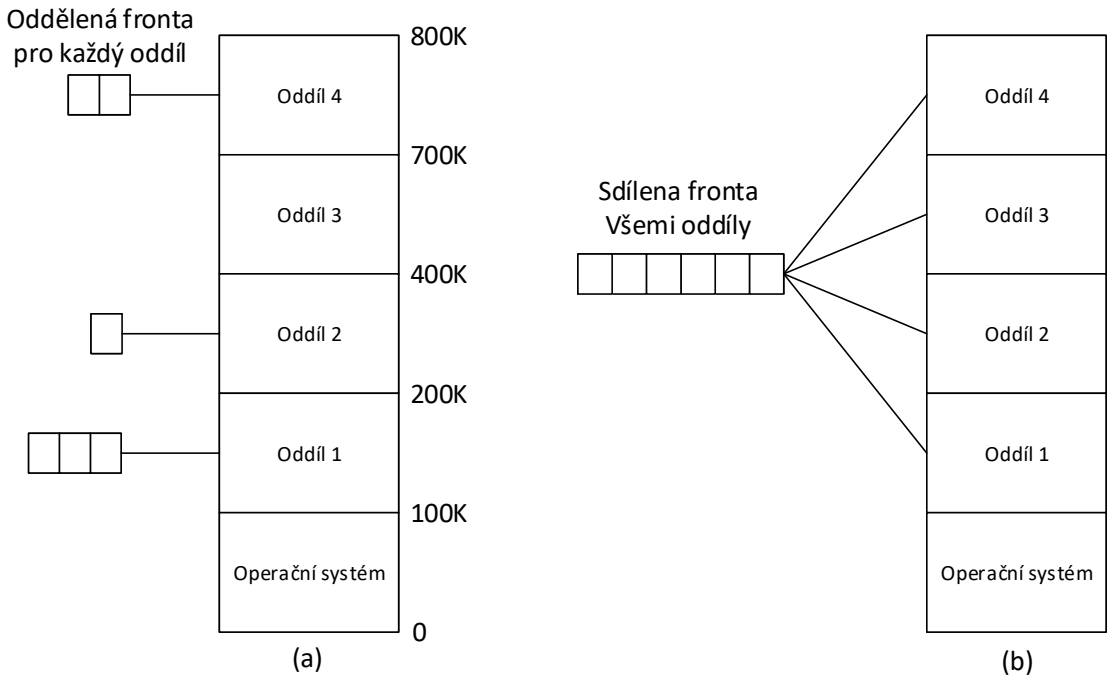
vyčkává na další vstupy od uživatele, podle kterých nahraje další program, kterým přepíše ten první a spustí jej. Protože tento systém neumožňuje běh více programů zároveň, tak se dnes již nepoužívá.

2.1.2 Více programový systém

Moderní systémy zvládají běh více programů zároveň. Toto umožňuje zvýšit utilizaci procesoru tím, že jakmile jeden proces čeká na vstupně výstupní operaci, některý z dalších běžících procesů může využít procesoru.

Takovéhoto systému můžeme dosáhnout například rozdělením paměti na *n* fixních částí. Bývají různě velké, abychom měli části pro malé i velké procesy. Procesy jsou vkládány do vstupní fronty nejmenší části paměti dostatečně velké, aby procesu stačila. S tímto přístupem může nastat plýtvání pamětí, když některá část zůstane nevyužita. Zatímco některé části paměti mohou být nevyužity, jiné části mohou mít několik čekajících procesů na vstupní frontě pro přiřazení paměti, přestože máme volnou paměť.

Řešením by mohlo být například použít pouze jedinou frontu pro všechny části paměti, a jakmile se uvolní část paměti, tak do ní přiřadit první proces ve frontě, pokud mu daná část paměti postačuje.



2.1: (a) Části paměti s oddělenými vstupními frontami pro každou část. (b) Části paměti se společnou vstupní frontou [1]

2.1.2.1 Relokace a ochrana paměti

Systém správy paměti s podporou více programů přináší dva zásadní problémy, které se musí řešit.

Relokace: Běžící program nezná a nemůže znát dopředu pozici, na které bude načten. To je problém pro používání instrukcí využívajících statické adresy (například skokové instrukce). Program musí používat adresy relativní k pozici kde byl načten. Kdyby například byl načten na adresu 100K a měl instrukci, která volá proceduru na adrese 100, je třeba, aby výsledná adresa byla 100K + 100.

Ochrana paměti: Mít v paměti více programů naráz vytváří riziko, že jeden program může zapsat do adresového prostoru jiného programu. Toto je nežádoucí a musí se tomu zabránit.

Jedna z možností, jak řešit relokaci je modifikace všech adres v binárním souboru při jeho nahrání. Tohle je však problematické z několika důvodů. Problém není jednoduše možné přesunout na jiné místo, vyžadovalo by to znovu přepočítat všechny adresy. Toto řešení neřeší problém ochrany paměti. Je možné vytvořit program, který si vygeneruje absolutní adresu za běhu a tu použije. Aby tohle řešení fungovalo, byla by potřeba vytvořit mapa všechny adres, které je potřeba upravit při nahrání.

Jedno možné řešení, které řeší jak relokaci, tak i ochranu paměti, je přidání dvou registrů zvaných **báze** a **limit**. Jakmile začne vykonávání procesu, do registru báze se nahraje adresa části paměti programu a do registru limit se nahraje délka této části paměti. Ke každé adrese použitou programem, je před jejím zavoláním přičten registr báze. Každá adresa je taky kontrolována, jestli nepřekračuje limit registr a tím se zajistí, že program může přistoupit mimo svoji část paměti. [2]

2.1.3 Swapování

Organizování paměti do několika fixních částí je jednoduché a efektivní pro dávkové systémy, dokud mají dostatek úkolů v paměti pro zaměstnání procesoru. Těm tento systém dostačuje.

Systémy se sdíleným časem a osobních počítačů již však tento systém nedostačuje. Nemusí být dostatek paměti, aby zvládla všechny aktivní procesy, proto přebytečné procesy jsou drženy na disku a načítány dynamicky podle potřeby.

Řešením tohoto problému je swapování (swapping). Swapování načte celý proces, který spustí a po nějaké době ho zase uloží zpátky na disk. Největší rozdíl mezi swapováním a systémem s fixní počtem částí pamětí je, že swapování může měnit počet lokaci a velikost částí pamětí dynamicky podle potřeby. Toto zvyšuje využití paměti, jelikož to není závislé

na fixních částech paměti a je to flexibilní. Nevýhodou je, že to zesložiťuje alokování a uvolňování paměti a sledování jejich stavu. [1]

2.1.4 Virtuální paměť

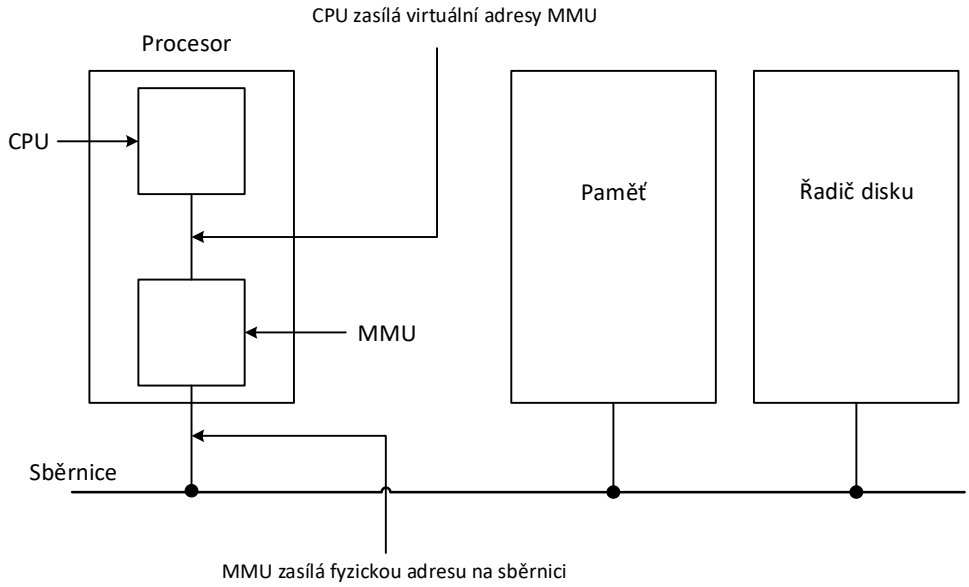
Dříve nastával problém, že se objevovaly programy, které byly moc velké na to, aby se vešly do dostupné paměti. Řešením bylo rozdělit program do několika částí. První část by začala jako první a po jejím skončení, by zavolala další část, tyto části byly uložené na disku a do paměti nahrávány dynamicky podle potřeby. Problémem bylo, že rozdělit program na části bylo na programátorech. Tudíž to přidělávalo práci navíc. [1]

Řešením toho problému je virtuální paměť. Základní myšlenka této metody je, že celková velikost programu, data a zásobníku mohou přesáhnout velikost dostupné fyzické paměti. V dobře navrženém systému virtuální paměti jsou v hlavní paměti pouze nejpoužívanější části procesového adresového prostoru. Ostatní části jsou uloženy na disku a načteny podle potřeby. Operační systém vytvoří iluzi jednotného bloku hlavní paměti. Toto je podpořeno hardwarovou podporou překladu virtuálních adres na fyzické adresy. Tento překlad se provádí na úrovni stránek a provádí ho MMU (Memory Management Unit). [3]

2.1.4.1 Stránkování

Většina systémů s virtuální pamětí využívá techniku zvanou stránkování. Adresy mohou být generovány s využitím indexování, bázových registrů, segmentových registrů a dalších.

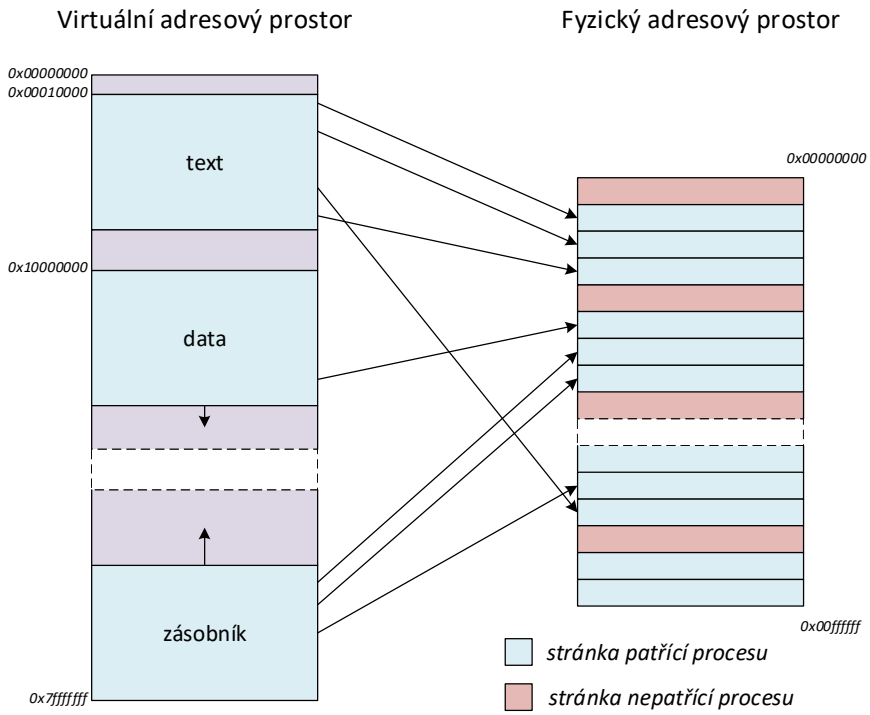
Tyto vygenerované adresy se nazývají virtuální adresy a tvoří virtuální adresový prostor. Rozdíl oproti systému bez virtuální paměti je ten, že na nich je adresa dána přímo na paměťovou sběrnici a na stejné fyzické adrese se bude číst či zapisovat. Zatímco systémy s virtuální paměti adresu prvně přeloží v MMU, které namapuje virtuální adresu na fyzickou, jak ukazuje obrázek 2.2.



2.2: Ukázka pozice a funkce MMU. MMU bývá běžně součástí CPU [1]

Paměť je organizována do částí se stejnou velikostí zvaných jako rámce stránky. K nim odpovídající části ve virtuální paměti jsou nazývané stránky. Stránky i rámce stránky mají stejnou velikost. Stránka slouží jako jednotka pro ukládání informací a také pro přenos mezi pamětí a diskem. Každý rámec stránky je identifikován adresou rámce což je adresa začátku rámce. [4]

Ukázka mapování mezi rámci stránky a stránkami je na obrázku 2.3. Všechny stránky nemusí být namapovány v případě, kdy by se program pokusil využít nemapovanou stránku by si toho MMU všimlo a CPU by vyvolalo trap. Tento trap se nazývá chyba strany (page fault). Operační systém za pomocí algoritmu pro nahrazení stránky vybere málo používaný rámec stránky a zapíše jeho obsah zpátky na disk. Pak nahraje právě odkazovanou stránku na uvolněný rámec stránky, upraví mapování a pokus se znovu provést instrukci, která vyvolala trap.



2.3: Ukázka mapování virtuálního adresového prostoru na fyzický adresový prostor [5]

2.1.4.2 Tabulky stránek

Virtuální adresa bývá většinou rozdělená na více částí například na číslo stránky a ofset. Kde číslo stránky je použito jako index do tabulky stránek pro naleznutí záznamu virtuální stránky. Ze záznamu stránky v tabulce je nalezeno číslo rámce stránky. Číslo rámce je přidáno k před ofset, kde nahradí číslo virtuální stránky. Tímto se získá fyzická adresa, která se může odeslat do paměti.

Účelem tabulky stránek je mapování virtuálních stránek na rámce stránek. O tabulce stránek by se dalo uvažovat jako o funkci, která přijímá číslo virtuální stránky jako argument a vrací číslo fyzického rámce jako výsledek.

Tabulky stránek čelí dvěma velkým problémům:

1. Tabulky stránek může být extrémně veliká
2. Mapování musí být rychlé

První bod vychází z faktu, že moderní počítače používají virtuální adresy minimálně 32 bitů dlouhé. Při velikosti stránky 4KB má 32 bitový adresový prostor 1 milión stránek. O 64 bitovém adresovém prostoru radši ani nemluvě. S 1 milionem stránek v adresovém prostoru musí i tabulka stránek mít 1 milión záznamů a každý proces musí mít svoji tabulku stránek (protože má svůj virtuální adresový prostor). [6]

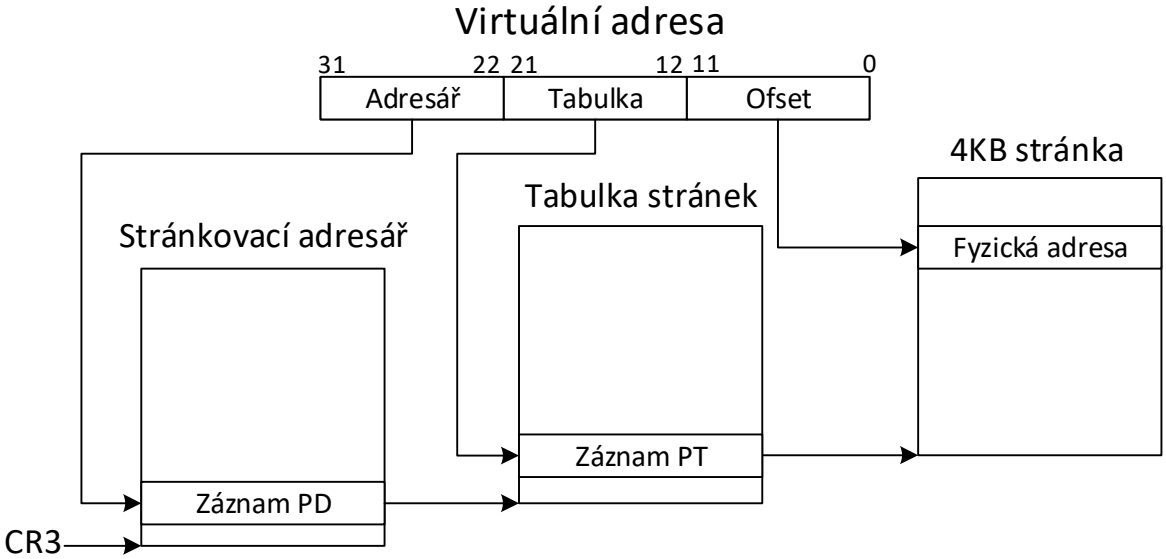
Druhý bod je důsledkem faktu, že mapování virtuální adresy na fyzickou se musí provést při každém přístupu do paměti. Některé instrukce mohou provést i několik přístupů do tabulky stránek. Pokud by přístup do tabulky stránek zabral značnou část času vykonání instrukce, mělo by to velký negativní dopad na výkon celého systému.

2.1.4.3 Víceúrovňové tabulky stránek

Řešením problému muset mít pořád v paměti obrovské tabulky stránek jsou víceúrovňové tabulky stránek. Ukázka víceúrovňové adresy stránek je na obrázku 2.4, na něm vidíme 32 bitovou adresu, která je rozdělená do tří částí, prvních 10 bitů na obrázku nazvané Adresář, značí index do tabulky stránek nejvyšší úrovně zvanou stránkovací adresář, dalších 10 bitů jsou Tabulka, které značí index do tabulky stránek druhé úrovně, posledních 12 bitů je ofset, který adresuje v rámci 4KB stránky.

Záznam získaný indexováním složky stránek obsahuje číslo stránky v tabulce stránek. Záznam získaný indexováním tabulky stránek obsahuje vlastní číslo rámce stránky pro danou stránku.

CR3 je registr na procesoru, který obsahuje ukazatel na stránkovací adresář.

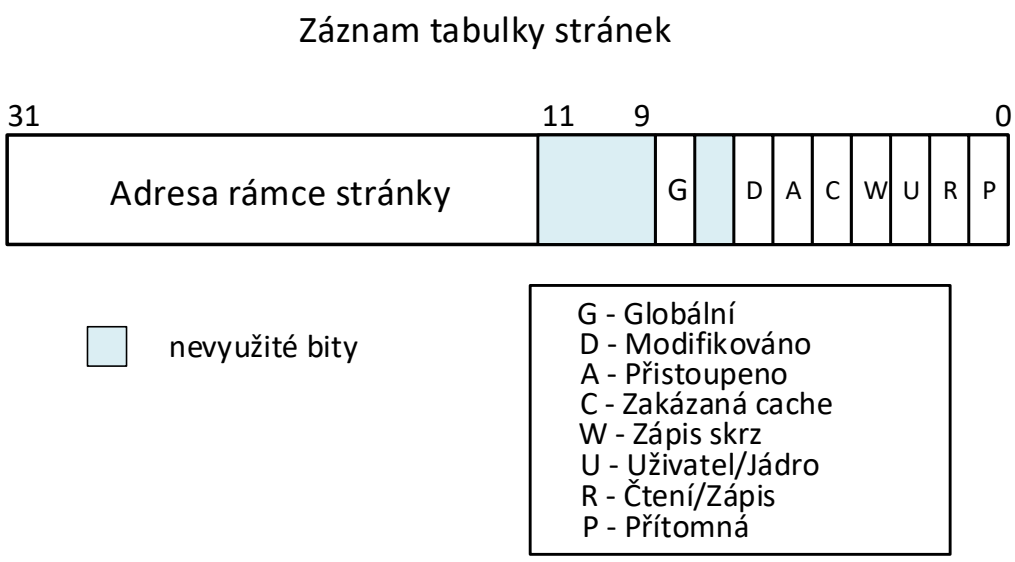


2.4: Ukázka překladu adresy přes víceúrovňové tabulky stránek na fyzickou adresu [7]

2.1.4.4 Struktura záznamu v tabulce stránek

Přesné rozložení záznamu je vysoce závislé na typu zařízení, ale obsažený druh informací je přibližně stejný na všech zařízeních. Na obrázku 2.5 je ukázka 32 bitové struktury záznamu tabulky stránky. Nejdůležitější částí je adresa fyzické stránky, přece jenom úkolem mapování stránky je získání této hodnoty. Dále tu máme bit značící přítomnost stránky. Pokud je bit

nastaven na 1, je záznam validní a může být použit. Jestli je bit nastaven na 0, virtuální stránka, které záznam náleží, není přítomná v paměti. Přístup k záznamu tabulky stránky s tímto bitem nastaveným na 0 způsobí výpadek stránky.



2.5: Ukázka struktury záznamu tabulky stránek [8]

Dále zde máme ochranný bit **R**, který říká, jaký druh přístupu je povolen. Pokud je nastaven na 1 pak se může číst i zapisovat, pokud je na 0 pak je možné záznam jenom číst.

Modifikováno (dirty) a přistoupeno (accessed) bity zaznamenávají užití stránky. Pokud je do stránky zapsáno hardware automaticky nastaví modifikováno bit. Tento bit je použit, když se operační systém využít záznam pro jinou stránku. Pokud stránka byla modifikována, musí být zapsána zpátky na disk. Jestli modifikována nebyla, tak může být prostě opuštěna, jelikož kopie na disku je stále validní.

Přistoupeno bit je nastaven kdykoliv je na stránku přistoupeno ať už čtením či zápisem. Jeho hodnota pomáhá operačnímu systému vybrat, kterou stránku zahodit v případě, že se objeví výpadek stránky. Stránky, které nejsou používané, jsou lepší kandidáti než ty, které jsou používané. Tento bit hraje důležitou roli u několika algoritmů pro nahrazování stránek popsaných později.

A nakonec bit zakázaná cache (cached) umožňuje vypínat caching. To je důležité pro stránky, které se mapují na registry zařízení místo paměti.

2.1.4.5 Struktura záznamu složky stránek

Struktura záznamu složky stránek je velice podobná struktuře záznamu tabulky stránek. Rozdílem je například, že místo adresy fyzické stránky odkazuje na adresu v tabulce stránek, že nemá bit modifikováno a má navíc bit značící velikost stránky, který když je nastaven bere se velikost stránky 4 MB a pokud není nastaven, bere se velikost stránky 4 KB.

2.1.5 Sdílená paměť

Sdílená paměť je část (segment) paměti ke kterému může přistupovat více procesů zároveň proto se tato část nazývá sdílená. Tato část je vytvořena jedním procesem a ostatní procesy se na ni mohou připojit a číst či zapisovat do ní.

Sdílená paměť se používá jako forma komunikace mezi procesy. Procesy si namapují vytvořený sdílený segment do svého adresového prostoru a poté mohou pomocí této paměti sdílet data s dalšími procesy. Sdílená paměť je pouze část paměti, a proto samy procesy, které ji využívají jsou zodpovědné za synchronizaci přístupu k ní. [9]

2.1.6 Translation Lookaside Buffers

Ve většině stránkovacích systémech tabulky stránek jsou udržovány v paměti, kvůli jejich velké velikosti. To může potencionálně mít velký vliv na výkon. Instrukce, která kopíruje jeden registr do druhého, může provést několik paměťových přístupů namísto jednoho, jak by se očekávalo.

Řešení tohoto problému je založené na postřehu, že většina program má tendence dělat velké množství přístupů do paměti na malém počtu stránek. Toto se nazývá lokalita referencí. Systémy využívají tohoto faktu pomocí TLB (translation lookaside buffer), což je malá paměť v MMU pro rychlé mapování virtuálních adres na fyzické bez nutnosti přistoupit k tabulce stránek. Obsahuje malý počet záznamů a připomíná odlehčenou verzi tabulky stránek. [10]

Jak MMU dostane virtuální adresu na překlad, nejdřív zkontroluje, jestli se její číslo virtuální stránky nenalézá v TLB, kde ho porovná se všemi záznamy. Pokud je nalezen vezme se rámec stránky z TLB, bez přístupu do tabulky stránek. Pokud není nalezen, tak ho MMU najde v tabulce stránek a nahradí jím záznam v TLB, aby při dalším přístupu ke stejné stránce byla v TLB nalezena.

2.1.7 Algoritmy pro nahrazení stránek

Algoritmy pro nahrazení stránek jsou techniky pomocí, který operační systém rozhodne a stránkách, které se mají odstranit, když je potřeba paměti pro další stránku. Toto se stane,

když nastane výpadek stránky a není žádná volná stránka nebo počet volných stránek je menší než požadovaný. Pokud je tato stránka znovu požadována musí se načíst z disku. [11]

I když by bylo možné vybrat náhodnou stránku odstranění při každém výpadku stránky. Výkon systému je výrazně lepší, když se odstraní stránka, která není moc využívaná. Pokud je odstraněna velmi využívaná stránka, bude pravděpodobně brzy znovu načtena, což by mělo za následek potřebu dalších operací.

2.1.7.1 Optimální algoritmus pro nahrazování stránek

Nejlepší možný algoritmus pro nahrazování stránek lze snadno popsat, ale je obtížné ho implementovat, jelikož vyžaduje znalost budoucích přístupů do paměti. Algoritmus by se dal shrnout jako: nahraďte stránku, která nebude použita nejdelší dobu. Použitím tohoto algoritmu bude nejmenší počet výpadků stránky. [12]

Teoretická implementace by byla udržovat u každé stránky číslo, které značí, kolik instrukcí proběhne, než se přistoupí k dané stránce. U některých to může být hned další instrukce u některých to může být až za tisíce instrukcí. Algoritmus vždy vybere stránku s nejvyšší hodnotou a tím oddálí výpadek stránky nejdále do budoucnost.

Přestože se tento algoritmus nedá realizovat využívá se pro porovnávání s dalšími algoritmy pro ukázání jejich efektivity.

2.1.7.2 Not Recently Used algoritmus pro nahrazování stránek

Tento algoritmus využívá dvou bitů, které se nachází u každého záznamu v tabulce stránek. Bity přistoupeno a modifikováno. Pomocí těchto bitů lze vytvořit jednoduchý algoritmus. Je potřeba tyto bity nastavit na 0 při vytvoření procesu a potom, periodicky nulovat bit přistoupeno, abychom rozeznali stránky, které nebyly nedávno využity.

Za využití těchto dvou bitů může při výpadku stránky operační systém rozdělit stránky do čtyř skupin:

0. Nepřistoupeno a nemodifikováno
1. Nepřistoupeno a modifikováno
2. Přistoupeno a nemodifikováno
3. Přistoupeno a modifikováno

Skupina číslo 1 nastane, pokud se u stránky, která je ve skupině číslo tři vynuluje bit přistoupeno pomocí periodického nulování.

Nepoužitý nedávno algoritmus odstraňuje náhodné stránky z nejnižší neprázdné třídy. Hlavní výhodou tohoto algoritmu je, že se dá lehce pochopit a jednoduše naimplementovat, i když jeho výkon není optimální. [1]

2.1.7.3 FIFO nahrazovací algoritmus

Dalším jednoduchým algoritmem je FIFO (first-in, first-out) algoritmus. Ten odstraňuje stránky, které byli v operační paměti nejdéle. Operační systém si udržuje seznam všech stránek, které jsou momentálně v paměti s tím, že na čele seznam je nejstarší stránka a stránka na konci je nejnověji přidaná stránka. Při výskytu výpadku stránky je stránka na čele odstraněna a nová stránka je přidána na konec seznamu. Problémem tohoto algoritmu je, že předpokládá, že nejstarší stránka bude nejméně potřebná, což ale nebývá vždy případem.

2.1.7.4 Nahrazovací algoritmus druhé šance

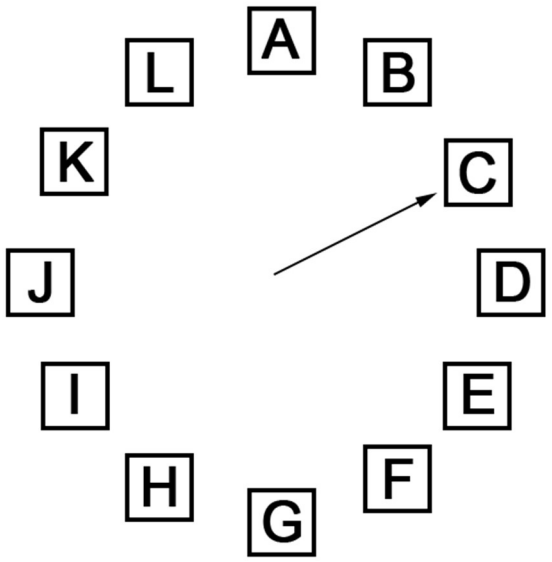
Jednoduchou úpravou FIFO nahrazovacího algoritmu se můžeme vyhnout problému zahazování používaných stránek. Stačí kontrolovat bit přistoupeno u nejstarší stránky. Pokud je 0 víme, že stránka je nejen stará, ale i nepoužívaná, taková to stránka se okamžitě nahrazuje. Pokud má bit přistoupeno nastaveno na 1, tak se byt vynuluje a stránka se přesune z čela na konec seznamu. [13]

Algoritmu se říká druhá šance proto, že stránku nevyhodí okamžitě, ale umožní znovu ji umístí do seznamu.

2.1.7.5 Hodinový algoritmus pro nahrazování stránek

I když je nahrazovací algoritmus druhé šance rozumný, je zbytečně neefektivní, protože neustále přesouvá stránky dokola na seznamu. Lepší přístup je mít všechny stránky na kruhovém seznamu ve formě hodin, jak ukázáno na obrázku 2.6. Ručička ukazuje na nejstarší stránku.

Když se vyskytne výpadek stránky, tak se stránka, na kterou ukazuje ručička, prohlédne. Jestli má nastaven přistoupeno bit na 0, tak je stránka odstraněna, nová stránka je vložena na její místo a ručička se posune o jedno místo. Pokud je přistoupeno bit nastaven na 1, tak je vynulován a ručička se posune na další pozici. Tento proces je opakován, dokud není nalezena stránka s přistoupeno bitem 0. Proto se tento algoritmus nazývá hodinový. Liší se od algoritmu druhé šance pouze implementací.



2.6: Ukázka kruhového seznamu pro hodinový algoritmus pro nahrazování stránek [1]

2.1.7.6 Least Recently Used algoritmus

Nejméně nedávno použité algoritmus pro nahrazování stránek nahrazuje stránky s nejmenší nedávnou aktivitou. Jelikož takové stránky mají malou pravděpodobnost, že budou brzo zase použity. Obdobně pokud je stránka používaná dá se očekávat, že bude používaná i v budoucnu. [14]

Přestože je tento algoritmus realizovatelný, není levný. Plně implementovat tento algoritmus by vyžadovalo udržovat seznam všech stránek v paměti s těmi nejvíce nedávno použitými na čele a nejméně nedávno na konci. Problémem je, že se tento seznam musí upravit při každém přístupu do paměti. Najít stránku, smazat ji a pak ji přesunout na začátek je náročná operace.

2.2 Webové technologie

Standartním jazykem pro webové dokumenty je HTML. Pro úpravu vzhledu se využívá kaskádových stylů CSS. Pro potřebu dynamických webů a možnosti reagovat na uživatelské akce se využívá skriptovací jazyk JavaScript. Jednotlivé technologie jsou detailněji popsány v následujících podkapitolách. Všechny tyto zdroje bývají uložené na serveru a klient v podobě webového prohlížeče si o ně žádá přes protokol HTTP.

HTML (HyperText Markup Language) [1] je značkovací jazyk pro vytváření hypertextových dokumentů. Jeho zápis vychází z jazyka SGML. Popisuje strukturu webových dokumentů za pomocí značek, které do sebe lze vnořovat. Každý značka má určité vlastnosti a vizuální podobu. Značky mohou mít přiřazené atributy, které mohou dále specifikovat vlastnosti elementu a jeho chování. Příkladem elementů může být obrázek, tučný text, odstavec či tabulka a další. Webový prohlížeč potom tyto značky a text jimi označený správně zobrazí. HTML se postupem času vyvíjí a momentálně je aktuální verzí HTML5. [15]

CSS[2] je jazyk, který popisuje vzhled dokumentu napsaného v HTML nebo XML. Umožňuje popsat pravidla, které určí, jak se vybrané elementy mají chovat a zobrazovat. CSS se skládá z dvojic selektor a deklaračního bloku. Selektor vybírá HTML elementy a deklarační blok popisuje jejich vlastnosti pomocí množiny pravidel ve formátu vlastnost:hodnota. CSS se dá zapisovat buď přímo do elementů HTML do jejich atributu ***style*** nebo vložit do HTML značky ***<style>***, která je pro to dělaná. Většinou se ale styly definují v odděleném souboru, který se dokumentu vloží pomocí značky ***<link>***. Aktuální verzí je CSS3. [15]

Javascript[3] je interpretovaných skriptovací jazyk. Přidává do dokumentu interaktivní element a dokáže dynamicky měnit obsah HTML dokumentu. Umí reagovat na různé události a uživatelské akce. Například na stisknutí tlačítka, přejetí myší, načtení dokumentu a další. JavaScript interpretují prohlížeče a do dokumentu se vloží buď do HTML značky ***<script>*** anebo je v oddělené souboru a vloží se do dokumentu stejně jako CSS. [16]

2.2.1 Jednostránková aplikace

Jednostránková aplikace (SPA – single page application) je extrémní přístup navrhování webových aplikací. Limituje strukturu aplikace tak, aby se celá vešla do jediné webové stránky, a omezuje další navigační schopnosti. Takové omezení silně ovlivňuje návrhový proces vývoje aplikace, především klade větší důraz na návrh uživatelského rozhraní. [17]

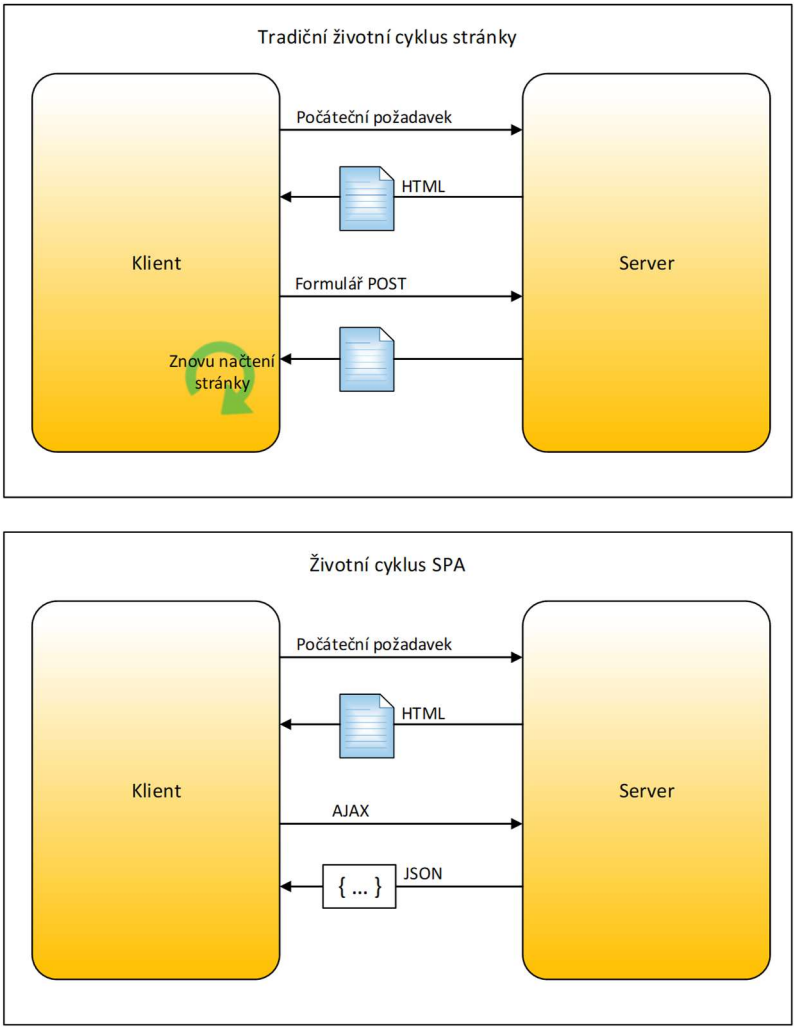
Takováto aplikace zobrazí jen jednu HTML stránku a dynamicky ji upravuje podle potřeby. Pro její tvorbu je potřeba JavaScript a AJAX (asynchronous JavaScript nad XML). JavaScript pomocí AJAX získává data ze serveru bez potřeby znovu načíst stránku, podle dat ze serveru pak upraví stránku. Výhodou je, že aplikace rychle reaguje na změny, protože nepotřebuje obnovit celou stránku.

---

[1] http://www.w3.org/html/

[2] http://www.w3.org/Style/CSS/specs

[3] http://www.w3.org/standards/webdesign/script



2.7: Porovnání životních cyklů tradiční a jednostránkové webové aplikace [18]

2.2.2 jQuery

jQuery[1] je nejrozšířenější JavaScriptová knihovna, která zjednodušuje skriptování webových stránek. Umožňuje manipulaci DOM (Document Object Model) což je stromová struktura reprezentující elementy webové stránky. jQuery zjednodušuje hledání a výběr těchto DOM elementů. S těmito vybranými elementy lze poté manipulovat. Krom toho jQuery umožňuje obsluhu událostí, tvorbu animací, AJAX požadavků a další věci. [16]

2.2.3 React

React je JavaScriptová knihovna vyvinutá facebookem pro tvorbu uživatelských rozhraní. Je založená na zapouzdřených komponentách, které si spravují vlastní stav. Z těchto komponent se poté poskládá komplexnější uživatelské prostředí. Navrhnou se jednoduché

[1] http://www.jquery.com

pohledy pro každý stav aplikace a React bude aktualizovat a vykreslovat potřebné komponenty. [19]

React mi umožní vytvořit uživatelské prostředí, které bude samo reagovat na nová data. Komponenty pro vizualizaci simulace stránkování budou hierarchicky zanořené v sobě a při změně dat, bude stačit je dát nejvyšší komponentě v hierarchii, ta potom bude propagovat data níže, dokud se nedostanou k relevantní komponentě, která se podle dat zaktualizuje, upraví stav nebo se vykreslí. Ideální pro tvorbu jednostránkových aplikací popsaných výše.[1]

2.2.4 AJAX

Asynchronní JavaScript a XML je technologie pro komunikaci klientské aplikace v jazyce JavaScript se serverem. Jedná se o komunikaci, kdy je vytvářen dodatečný požadavek od klientské aplikace k serveru. Jeden z požadavků můžu být například přístup k datům v databázi. Data, která chce server odeslat jako odpověď na dotaz, jsou nejčastěji formátována do formátu JSON, protože ho JavaScript dokáže dekódovat zpět na data, kterým rozumí. [20]

2.2.5 JSON

JSON je jednoduchá formát pro výměnu dat, lehký na čtení u zápis, oproti XML je jednodušší na generování a zpracování. Je založen na podmnožině JavaScriptu, využívá textového formátu nezávislého od programovacího jazyku. Tyto vlastnosti mají za následek, že JSON se stal populárním formátem pro výměnu dat.

JSON popisuje datové objekty ve dvou formách: polích a objektech. Pole v JSON je seřazená množina hodnot. Začíná s `"["` a končí s `"]"`, a hodnoty jsou v poli oddělené pomocí čárky. Objekty je neseřazená množina jméno/hodnota dvojic. Objekt v JSON začíná s `"{"` a končí s `"}"`, každé jméno je následováno `":"` a každá dvojice je oddělena čárkou. [21]

2.2.6 WebSockety

WebSockety je nová technologie v HTML 5, která zprostředkovává interaktivní komunikaci mezi prohlížečem a serverem. Umožňuje serveru zaslat správu a přijmout odpověď bez toho, aby se muselo opakovaně dotazovat na serveru. [22]

[1] https://facebook.github.io/react/

2.2.7 Scalable Vector Graphics

Scalable Vector Graphics je standart, který popisuje 2D grafiku, která může být interaktivní a animovaná. Formát SVG je přenositelný, založen na XML a má dokumentově orientovanou architekturu.

SVG umožňuje vykreslovat všechny základní vektorové tvary jako jsou čtverce, obdélníky, trojúhelníky a kruhy. Kromě těchto základních tvarů umí SVG vykreslit i složitější obrazce jako jsou Bézierovy křivky a eliptické oblouky. Na všechny grafické objekty umožňuje SVG aplikovat transformace (rotace, translace, škálování, zkosení).

SVG podporuje také animace, stačí si vybrat nějaký XML element nastavit atributy či CSS styly a pak definovat čas startu a délku trvání.

I když je SVG převážně pro vektorovou grafiku, tak podporuje i rastrové obrázky, které se dají vkládat a kombinovat s dalšími objekty v SVG formátu. [23]

2.2.8 Bootstrap

Bootstrap je sada stylů a JavaScript souborů, které pomáhají při tvorbě webových stránek. Obsahuje velké množství výchozích stylů pro standartní prvky stránky jako jsou tlačítka, nadpisy, formuláře a jiné. Toto urychluje vývoj stránek, jelikož nemusíme psát vlastní styly, ale můžeme využít již definované styly. Bootstrap se stará o to, aby jeho styly vypadaly stejně ve všech prohlížečích. Bootstrap umožňuje tvorbu responzivních webů, což znamená že se webové stránky přizpůsobují podle velikosti obrazovky zařízení tak, aby efektivně využily daný prostor. [24]

2.2.9 Selenium

Selenium[1] je nástroj, který umožňuje automatizaci webový prohlížečů. Takto se dají automatizovat různé úkony vykonávané na webu, jeho hlavní využití však je na tvoření testů pro webové aplikace. Kdy umožňuje vytvářet akce v prohlížeči, které simulují aktivitu uživatele na webové stránce a pak kontroluje, zda stránka reaguje, jak má.

2.2.10 NodeJS a NPM

Platforma NodeJS je běhové prostředí pro vytváření a běh aplikací. Aplikace se píší v JavaScriptu což je výhodou, protože pak může vývojář mít jak serverovou, tak klientskou aplikaci napsanou ve stejném jazyce. Prostředí lze použít pro vytváření serverových aplikací či pro běh lokálních aplikací.

[1] http://www.seleniumhq.org/

Podporuje komunikaci přes TCP protokol, práci se sockety, práci se soubory a mnoho dalšího. Platforma je rozšiřitelná a existuje pro ni obrovské množství modulů. Tyto moduly jsou přístupné přes správce balíčků NPM. Tento správce je v základu přidáván při instalaci NodeJS. Balíčky se dají lehce instalovat za pomocí příkazů v konzoli. Příkaz **`npm install [název_balíčku]`** slouží pro nainstalování balíčku. Aby se nemusel každý balíček instalovat ručně bývá běžné, že se vytvoří soubor **package.json**, který v sobě obsahuje informace o projektu a seznam balíčků na kterých je projekt závislý.

Všechny balíčky uvedené v tomto souboru lze poté nainstalovat všechny naráz pomocí příkazu **`npm install`**. Stažené balíčky se ukládají do adresáře node_modules, která se vytvoří v adresáři projektu. [25]

2.3 Linuxové kontejnery

Linuxový kontejner je množina procesů, které jsou izolované od zbytku systému. Každý kontejner vychází z nějakého obrazu, který obsahuje závislosti aplikace, takže je kontejner velice přenositelný a výhodný pro vývoj.

Je možno mít spoustu kontejneru každý s jinou konfigurací a mít tak, vývojový kontejner, testovací kontejner či produkční kontejner. Díky veliké možnosti konfigurace jsou kontejnery velice flexibilní a dají se použít na spoustu věcí.

Na rozdíl od virtualizace kontejnery nepouštějí celé operační systémy místo toho sdílí hostující operační systém a jen od něj izolují aplikační procesy. Díky tomu je kontejner výrazně lehčí a rychlejší, jak virtualizace.

Další výhoda a možné využití kontejneru je pro testování nebezpečného kódu. Jelikož je kontejner izolován od zbytku systému není možné pro aplikace v něm, aby shodily či poškodily hostitelský systém. Toto se hodí pro testování. Pokud by aplikace poškodila kontejner, tak se dá lehce obnovit ze zálohy.

Kontejnery lze také omezovat jako například kolik paměti, či místa na disku mají lze i omezit CPU výkon. Tohoto se dá využít buď pro správu zdrojů mezi kontejnery nebo pro testování v neoptimálních podmínkách. [26]

Kapitola 3

Návrh aplikace

V této kapitole budou popsány požadavky aplikace, jaké funkce by měla zvládat a jak by je měla provádět. Je tu popsán návrh jak aplikace, tak konfigurátoru, také jsou tu vysvětlená návrhová rozhodnutí, která jsem během návrhu učinil.

3.1 Požadavky aplikace

Aplikace by měla sloužit pro výuku stránkování a paměťového systému se zaměřením na praktické vyzkoušení některých konceptů paměťového systému. Aplikace by měla být schopná simulovat stránkování a tuto simulaci vizualizovat, tak aby uživatel viděl, jak stránkování probíhá a funguje. K tomu by aplikace měla umožňovat uživateli napsat část funkcionality, napsat algoritmus či ošetřit nějaké stavy. Aplikace by poté měla vzít tento kód a pokusit se simulovat stránkování s tímto kódem, jestli je kód špatný, tak ukázat chybu, pokud je kód funkční, pak by měl uživatel vidět na vizualizaci, co jeho kód provádí a zda funguje, jak má.

Dalším požadavkem bylo, aby uživatel mohl do jisté míry simulaci ovládat a procházet ji po menších krocích, kde by uživatel mohl koukat na stav systému a hodnoty tempem, které mu vyhovuje.

3.2 Analýza požadavků

Jelikož přímo v zadání bylo definováno, že aplikace měla být vyvinuta jako webová, bylo jasné, že uživatel bude používat jakousi webovou aplikaci, která bude prezentovat informace a přijímat vstupy. Větší problém byl s tím, že aplikace měla umožnit psát části uživateli kódu.

Je jasné, že tento kód bude potřeba přeložit a získat nějaké výstupy či ho řádně otestovat. Jako jazyk, který bude požadován po uživateli, jsem zvolil jazyk C. Za prvé je to jazyk, se kterým mají zkušenosti studenti FIT VUT, zadruhé je to jazyk nejběžnější používaný pro programování operačních systémů a v rámci nich paměťových subsystémů. Takže bylo zapotřebí najít způsob, jak překládat či interpretovat jazyk C v rámci webové aplikace. To se ukázalo jako docela problém, jelikož C je kompilovaný jazyk a webový prohlížeč má z důvodu bezpečnosti spoustu omezení, co může JavaScriptový kód dělat.

Nemůže například sám vytvářet a používat soubory, stejně tak není možné pouštět binární soubory.

Přesto jsem se pokoušel najít nějaký nástroj, který by překládal kód jazyka C v prohlížeči. Nalezl jsem jeden nástroj, který toto uměl a původně jsem jej chtěl využít, ale po experimentaci s ním jsem objevil velké nedostatky, hlavní z nich byl, že nástroj nepodporoval uživatelem definované datové struktury.

Jako další možné řešení mě napadlo naimplementovat interpretátor jazyka C v JavaScriptu sám. Naleznul jsem nástroje jako je například Jison[1], které umožňují generování syntaktického analyzátoru v JavaScriptu, ale vzhledem k omezením webových prohlížečů popsaných výše a nemožnosti udělat některé věci, jako je například zarovnávání adres, které se využívá u stránkování k tomu, aby se uvolnily některé bity adresy pro flagy, jsem si řekl, že nejlepší bude využít existující překladač.

Zbývalo přijít na to, jak vytvořit spolupráci mezi webovou aplikací a překladačem. Jako jediné řešení mě napadlo vytvořit program, který by se choval jako server a komunikoval s webovou aplikací. Webová aplikace by mu posílala kód, server by tento kód vzal, přeložil jej pomocí překladače za použití příkazů shellu a odeslal výstup či chybové hlášení serveru. Toto řešení má nevýhodu v tom, že je za potřebí server a mám menší kontrolu nad překladem. Také webová aplikace bude potřebovat nastavit adresu serveru pro její správný chod. Výhodou je, že budu používat kvalitní a prověřený překladač, který zvláště při kompilačních chybách bude generovat kvalitní chybové hlášky, na které už je většina uživatelů zvyklá. Další výhodou je, že budu za pomoci serveru používat i další nástroje, které nabízí shell, na které bude server spuštěn, a také budu mít možnost vytvářet, mazat a používat soubory, což zjednoduší práci s knihovnami.

Jelikož uživatel bude mít možnost vytvářet části kódu, bylo jasné, že bude zapotřebí mu nabídnout již vytvořené datové struktury a funkce pro práci s nimi z důvodu, aby se mohl uživatel soustředit na úzce zaměřenou část či aspekt stránkování a nemusel sám napsat celý paměťový systém. Z tohoto důvodu bude potřeba vytvořit nějaké knihovny, které implementují důležité části paměťového systému potřebné pro stránkování. Ty by uživatel využíval při psaní svého kódu.

Uživatel také bude očekávat nějakou reakci na jeho kód, zda je v pořádku a zda funguje, jak má. Pro kontrolu uživatelského kódu jsem se rozhodl použít testy. Pro každou funkci, kterou bude uživatel moci implementovat, bude vytvořen testový soubor, který vezme uživatelský kód a pustí na něm sadu testů, které zkouší jeho funkcionalitu. Každý z těchto testů vygeneruje výstup, zda uspěl či neuspěl s krátkým popisem a toto server odešle zpátky webové aplikaci, aby mohla výsledky prezentovat uživateli.

[1] http://zaa.ch/jison/

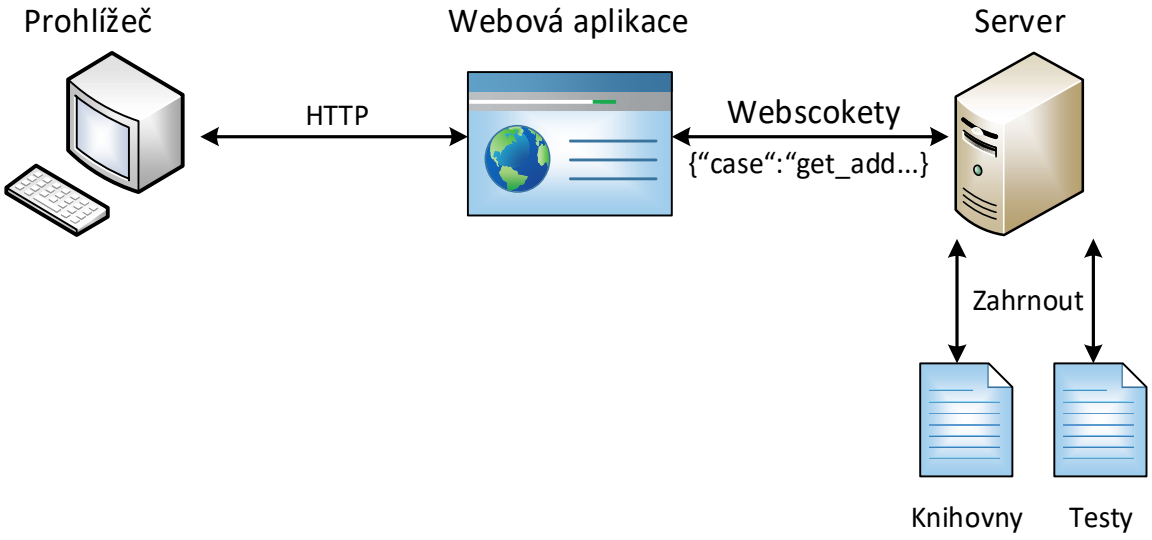
Před tím, než bude umožněno uživateli otestovat jeho kód, bude muset zkusit svůj kód zkompilovat, aby se ujistil, že se dá tento kód přeložit. Pokud při překladu nastanou chyby, server tyto chyby od překladače pošle webové aplikaci a ta je zobrazí uživateli, aby věděl, kde udělal chybu a mohl si ji opravit.

Další věc, kterou je potřeba vymyslet, je, jak uživateli zobrazit práci stránkovacího systému. Tabulka stránek prochází různými fázemi během operací, jako jsou například vyhledání indexu adresáře stránek, nalezení adresy tabulky stránek, vytvoření tabulky stránek a jiné. Tyto fáze a operace si představuji jako stavy a přechody mezi nimi. Pro každý takovýto stav bude vytvořena vizualizace či obrázek a za pomocí stavového automatu ovládaného webovou aplikací se bude přepínat mezi těmito stavy, aby bylo znázorněno, jak aplikace za sebou probíhají a co se při tom děje s tabulkou stránek.

3.3 Návrh systému

Jak bylo odůvodněno výše, rozhodl jsem se aplikaci rozdělit do více částí. Webová aplikace, které bude obstarávat interakci s uživatelem a bude působit jako takzvaný frontend a server, který bude pracovat s knihovnami, testy a uživatelským kódem a bude reprezentovat backend systému.

Tyto části budou spolupracovat a komunikovat, jak je znázorněno na obrázku 3.1. Níže jsou jednotlivé části rozepsány detailně.



3.1: Ukázka částí systému a komunikace mezi nimi

3.3.1 Webová aplikace

Webová aplikace bude mít na starosti prezentaci konceptů uživateli, uživatel si bude moci vybrat některý z konceptů, kde by si mohl spustit simulaci a dozvědět se informace o daném

konceptu. Dále jedna z hlavních věcí, co bude webová aplikace dělat, je vizualizace stránkování a paměťového systému, tato vizualizace bude zobrazovat stavy, ve kterých se budou jednotlivé části simulovaného paměťového systému nacházet, uložené hodnoty například v tabulkách stránek či adresář stránek. Tuto vizualizaci bude aplikace umožňovat ovládat a to tak, že buď vizualizace pojede sama nějakou rychlostí, nebo bude uživatel sám krok po kroku procházet stavy simulace.

Krom toho bude webová aplikace zodpovědná za přijímání uživatelských vstupů, především pak umožní uživateli psát kód v jazyce C, aplikace tak bude obsahovat komponentu, která umožní uživateli psát kód. Pro pomoc psaní bude tato komponenta mít automatické odsazování, barevné zvýrazňování.

Každá část, kterou bude moci uživatel implementovat, bude reprezentovaná jako jedna stránka. Každá takováto stránka bude obsahovat popis konceptu a také co se očekává od uživatele na naprogramování. Pak bude zobrazená komponenta, kam bude uživatel moci zapisovat svůj kód. Tato komponenta již bude mít základní kostru funkce, jejíž součástí bude signatura funkce, aby uživatel viděl, jaké parametry funkce přijímá a co má funkce vracet. Součástí této kostry také budou komentáře, které budou sloužit jako jakýsi návod uživateli pro to, jak danou funkci implementovat. Budou radit například s čím začít, co by měl vracet či na co by si měl dávat pozor. Tato stránka také bude zobrazovat funkce a datové struktury, které má uživatel k dispozici pro použití ve svém kódu. Tento výpis funkcí bude pro každý případ jiný a bude obsahovat funkce specifické pro danou funkci. Většina těchto funkcí bude volitelných a nebude nutné je využívat, ale některé mohou být i povinné na použití v dané funkci. Tyto funkce budou abstrahovat některé části paměťového systému.

Poslední zodpovědností bude komunikace se serverem, kterému bude zasílat uživatelem napsaný kód a nějaké informace k tomu, jako například jaká funkce se zrovna implementovala či jestli se chce jen překlad nebo i testování. A bude přijímat výstupy ze serveru. Především pak výstup překladače, aby uživateli zobrazila chyby, které měl ve svém kódu, nebo naopak potvrdila, že jeho kód lze přeložit a poté výstupy testů, aby uživateli zobrazila, jaké testy prošly a jaké neprošly.

3.3.2 Server

Server bude desktopová aplikace pro Linux, která bude pracovat se shellem a využívat jeho nástrojů, tato aplikace bude poslouchat na definovaném portu a spustí obsluhu pro každé připojení, které obdrží, bude tak moci obsluhovat více webových aplikací zároveň.

Hlavní funkcí serveru bude překládání kódu zaslaného z webové aplikace, aby tento kód mohl být přeložen, nejprve se do něj vloží hlavičkové soubory použitých knihoven, které

přidají do kódu funkce a struktury, které může uživatel ve svém kódu využívat a přidá main funkci. Aby server věděl, které hlavičkové soubory přidat, co s kódem dělat a jaké testy využít, bude webová aplikace posílat kromě kódu i informace, o jakou funkci se jedná a zda jej chce jen zkompilovat či otestovat.

V případě pouze kompilace se kód přeloží pomocí překladače a výstup překladače se odešle webové aplikaci. Pokud překladač objeví chyby, tyto chyby server zpracuje a pošle o nich informaci zpátky webové aplikaci, která kód zaslala. Pokud při překladu nenastane žádná chyba, pak zašle zprávu o úspěšném přeložení.

Pokud bude webová aplikace požadovat testování, přiloží server k uživatelskému kódu i testovací kód k dané funkci. Tento celek přeloží a poté spustí výsledný binární soubor a získá jeho výstup. Tento výstup obsahuje informace o tom, jak dopadly testy, a server tyto informace odešle serveru.

Krom tohoto bude mít server ještě zodpovědnost za kontrolu kódu, zda nepoužívá některé nepovolené funkce, jako je alokování paměti nebo tvorba či mazání souborů z bezpečnostních důvodů.

3.3.3 Knihovny

Další částí tohoto sytému jsou knihovny napsané v jazyce C. Tyto knihovny budou obsahovat funkcionalitu paměťového systému jako například TLB, MMU, funkce pro sdílení paměti, tabulky stránek a jiné. Účelem těchto knihoven je poskytnutí funkcí a datových struktur uživateli, který píše kód na webové aplikaci, aby se mohl úzce soustředit na daný koncept a nemusel implementovat celý paměťový systém.

Knihovny budou ve složce spolu se serverem. Knihovny budou mít hlavičkové soubory, za pomoci kterých, bude server přidávat jejich funkcionalitu k uživatelskému kódu.

3.3.4 Testy

Další částí tohoto systému jsou testy. Každá funkce, kterou bude uživatel moci napsat ve webové aplikaci, bude mít napsanou sadu testů. Každý tento soubor bude obsahovat minimálně jeden test. Testů bude tolik, aby pokryly všechny hlavní operace, které by měla funkce vykonat. Každý test bude vracet výstup o úspěchu testu se stručným popisem testu.

3.3.5 Návrh webové aplikace

Webová aplikace bude napsána za použití standartních webových technologií jako je HTML, CSS a JavaScript. Případně budou využity nástroje zjednodušující psaní v těchto technologiích.

Aplikace bude prezentovat několik konceptů z paměťového systému. Tyto koncepty jsou získání fyzické adresy z virtuální adresy, mapování stránky, obsluha výpadku stránky a sdílení paměti.

Každá stránka bude umožňovat uživateli napsat jednu či více funkcí podle potřeby. Její zodpovědnost bude prezentovat uživateli daný koncept a pomoci mu v implementování funkce na daný koncept. Pro tento účel bude stránka pro každou funkci obsahovat textový vstup, kam uživatel bude psát kód. Každý tento textový vstup bude mít k dispozici dvě tlačítka, tlačítko kompilovat a tlačítko otestovat. Webová aplikace podle tlačítka rozhodne, či odeslat kód pouze na překlad nebo jej i testovat.

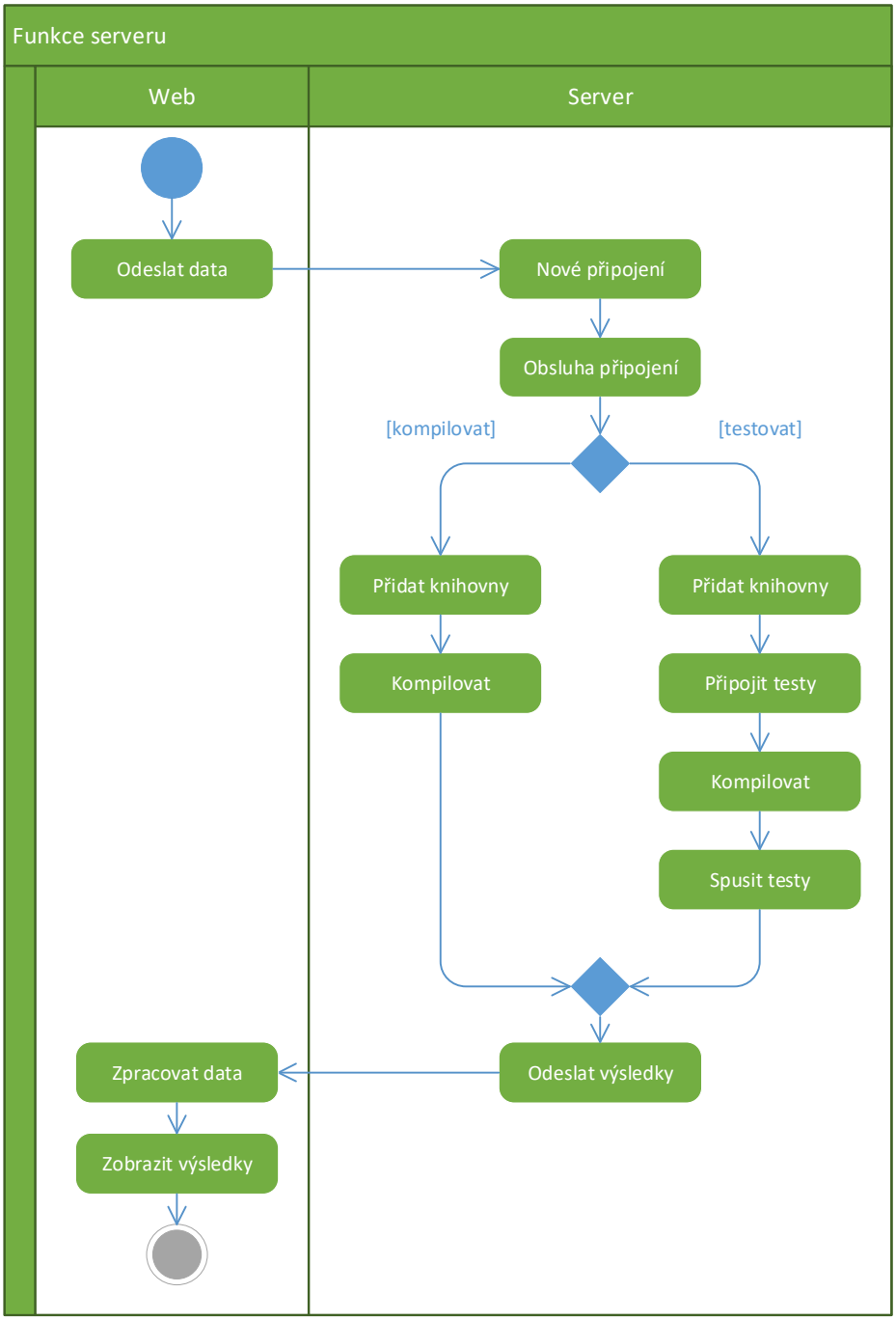
Webová aplikace bude též zodpovědná za navázání komunikace se serverem, kterému bude odesílat kód a potřebné informace pro jeho práci. Poté bude interpretovat výsledky odeslané serverem. Výsledky zobrazí uživateli, kterému tak dá odezvu na jeho vstup.

Další zodpovědností aplikace bude vizualizace stránkování. Pro tento účel bude implementovat stavový automat, který bude podle potřeby vhodně zobrazovat tabulky stránek a operace, které se tam odehrávají.

3.3.6 Návrh serveru

Server bude jakýsi zprostředkovatel linuxových nástrojů, jako je překladač webové aplikaci. Bude to program, který neustále poběží jako server a bude na portu očekávat komunikaci, pro každý požadavek spustí obsluhu podle toho, jaký požadavek to je, jestli kompilace či testování, a přidá správné hlavičkové soubory či testy.

Kromě testování a kompilace bude také jeho úkolem po sobě uklízet, což znamená mazat všechny vygenerované soubory při kompilaci, jakmile již nejsou potřeba. Hlavní aktivity, které bude server provádět, jsou zobrazeny na obrázku 3.2.

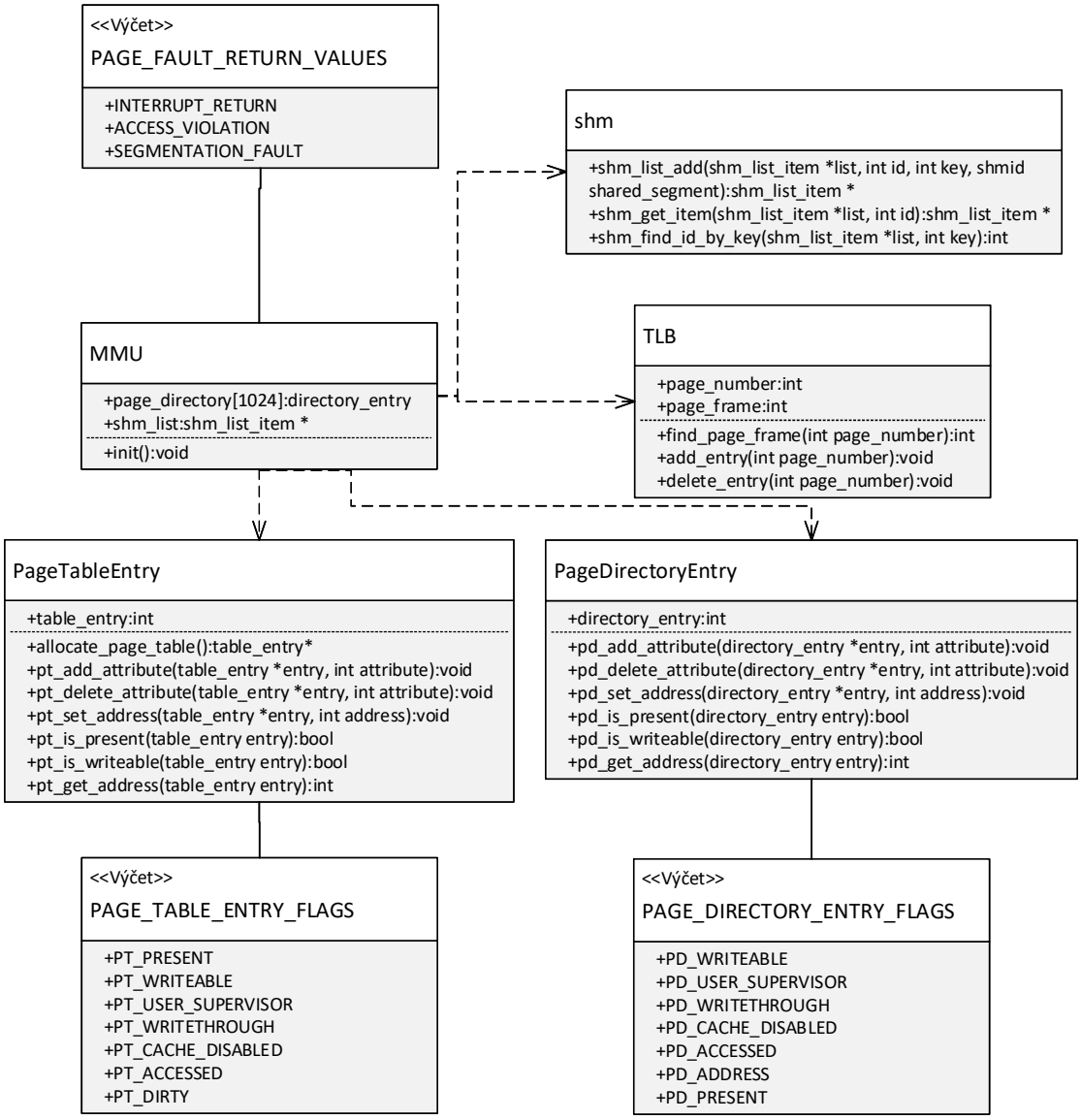


3.2: Ukázka zobrazující aktivity serveru a webové aplikace

3.3.7 Návrh knihoven

Knihovny budou obsahovat datové struktury a funkce, které bude uživatel využívat pro psaní svého kódu. Tyto knihovny spolu s uživatelským kódem vytvoří model paměťového systému se stránkováním, který bude poté testován. Knihovny budou napsány v jazyce C a k uživatelskému kódu budou přidány hlavičkové soubory těchto knihoven.

Příklad některých knihoven, které budou vytvořeny, a jaké základní funkce budou poskytovat, je znázorněno diagramem na obrázku 3.3



3.3: Třídní model knihoven

Kapitola 4

Implementace systému

V této kapitole je pospaná implementace této diplomové práce. Je zde popsaná každá část systému, takže webová aplikace, server, testy, knihovny. Také je tu popsán protokol a způsob, jakým spolu tyto části komunikují. Také tu bude popsáno odůvodnění, proč jsou jednotlivé části implementovány tak, jak jsou. Část bude věnována také tomu, jaké nástroje a technologie byly využity pro implementaci, jak tuto aplikaci zprovoznit a jaké má minimální požadavky a závislosti.

4.1 Vývoj webové aplikace

Webová aplikace byla první část, na které jsem pracoval, spolu se serverem. Ze začátku pouze primitivní stránka bez ničeho s jediným formulářem, která sloužila jako jakýsi prototyp na otestování komunikace za pomocí websocketů. Během práce na této diplomové práci prošla webová aplikace asi největšími změnami.

Ze začátku jsem celou webovou aplikaci a rozhraní implementoval za pomocí Reactu jako SPA (Single Page Application). Toto se neukázalo jako velice vhodné řešení, jelikož React je vhodný hlavně na zobrazování dat a jejich změn, jelikož komunikace se serverem v té době byla velice jednoduchá a nebylo moc dat, která se navíc neměnila, nevyužíval jsem nijak benefitů této technologie a psaní webové stránky za pomoci této technologie mi zabíralo výrazně více času, než by mi zabralo to napsat bez Reactu. A jelikož se mi začala líbit myšlenka mít každý koncept stránkovacího systému odděleně, přepsal jsem webovou aplikaci čistě v HTML, CSS a Javascriptu na několik stránek. Pracovat na těchto stránkách bylo výrazně jednodušší a efektivnější.

Nakonec jsem ale všechny stránky přece jen spojil do jediné z několika důvodů. Za prvé si byly všechny tyto stránky velice podobné a obsahovaly obdobnou funkcionalitu, což vedlo k poměrně velké duplikaci podobného kódu na všech stránkách. Za druhé jsem často dělal stejnou změnu v několika souborech, také jsem chtěl sdílet některé informace, konkrétně IP adresu a port serveru, mezi všemi stránkami tak, aby při konfiguraci stránky stačilo nastavit adresu na jediném místě. Toto navíc způsobilo, že jednotlivé případy si zachovávaly všechen kód a stav, který byl zadán uživatelem a není potřeba čekat na načítání při kliknutí na odkaz na jiný případ.

Takže ve výsledku byla webová aplikace implementovaná jako jediná stránka. Jednotlivé případy (získání fyzické adresy, mapování stránek, obsluha výpadku stránky, sdílení paměti) se při kliknutí na odkaz dynamicky skrývají či zobrazují podle potřeby.

4.2 Implementace webové aplikace

Všechny případy mají stejné rozložení stránky. V horní části stránky je menu, kde si může uživatel přepínat mezi případy. Pak následuje obrázek či vizualizace relevantní k danému případu. Následuje název případu, jeho popis a co se očekává od uživatele za funkci.

Pod ním se nachází textový formulář, kde uživatel zadává svůj kód a řešení funkce. Tento formulář je předvyplněný signaturou funkce a pár komentáři, které pomáhají nasměrovat uživatele správným směrem. Pod každým formulářem jsou dvě tlačítka jedno pro odeslání kódu pro kompilaci a druhé pro odeslání kódu na testování. Na konci každé stránky se pak nachází výpis funkcí, výčtových typů či struktury, které uživatel může pro daný případ využít. Tyto výpisy obsahují kromě signatur funkcí i krátký popis, k čemu daná funkce slouží. Některé obecné funkce jsou ve výpisech všech případu, ale některé funkce mohou být specifické pouze pro daný případ. Některé funkce mohou vyžadovat, aby je uživatel využil ve svém kódu, jinak by jeho kód nemusel projít testy, takovéto funkce to většinou mají u sebe zdůrazněné. Návrh použitého rozložení stránky při případu je zobrazen na obrázku 4.1.

4.3 Popis případů

Celkem jsem vytvořil čtyři případy, které se soustředí na některé koncepty paměťového systému. Případem nazývám jednu stránku, které se zabírá jedním konceptem, u kterého se očekává, že uživatel naprogramuje část funkcionality. Konkrétně se soustředí na manipulaci s tabulkou stránek, sdílení paměti a obsluze výpadku stránky.

4.3.1 Získání fyzické adresy

První případ je získání fyzické adresy z virtuální. Toto je jednoduchý případ, jehož úkolem je za pomocí dvojúrovňové tabulky stránek získat fyzickou adresu z virtuální. Úkolem uživatele aplikace je správně získat indexy z virtuální adresy za pomocí těchto indexů vybrat v adresáři stránek správnou tabulku stránek a z ní správnou stránku, ze které se přidá adresa po přidání ofsetu získá uživatel fyzickou adresu. Toto je funkce, kterou vykonává paměťová jednotka procesoru automaticky a která se děje při každém přístupu k paměti, tak je dobré ji znát.

4.3.2 Mapování stránky

Druhý případ řeší mapování fyzické adresy na virtuální. Začíná obdobně jak předchozí případ, kdy se získají indexy do tabulky stránek. Akorát v tomto případě nemusí daná tabulka existovat v tom případě je potřeba ji vytvořit. A poté najít ten správný záznam a

do něj přiřadit fyzickou adresu. Tento případ demonstruje, jak je možné namapovat fyzickou adresu do virtuálního adresového prostoru.

4.3.3 Obsluha výpadku stránky

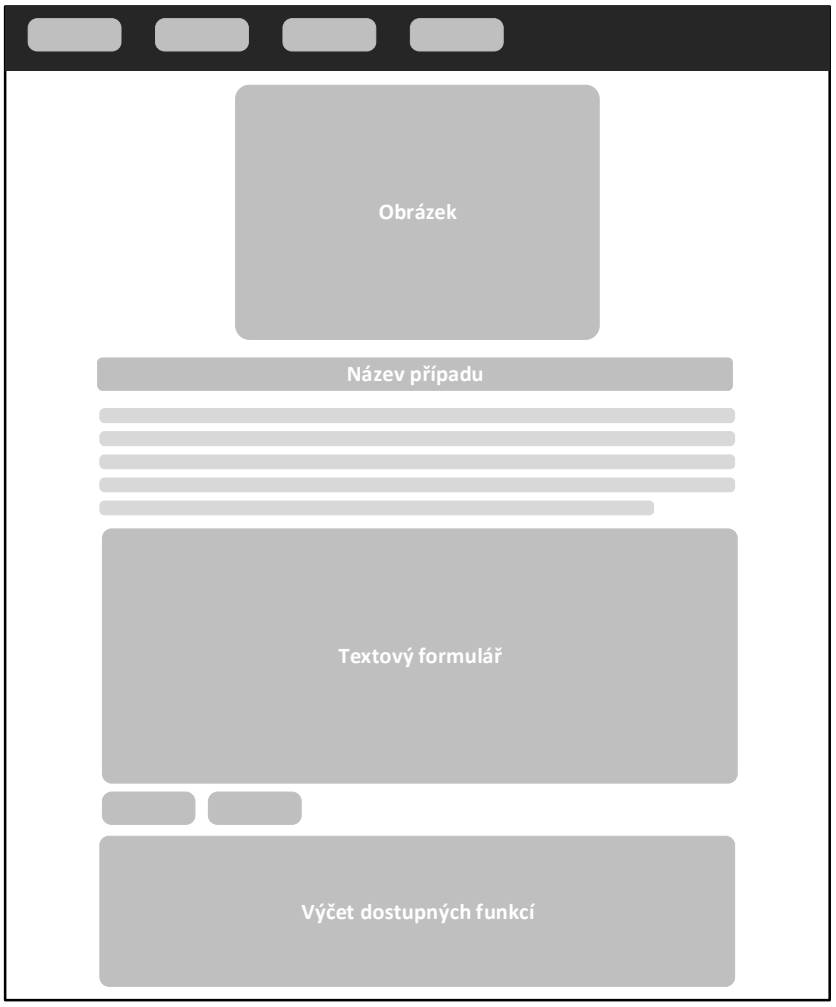
Tento případ řeší, jak by se měl obsloužit výpadek stránky. Od uživatele se očekává funkce, která na základě adresy, která způsobila výpadek a chybového kódu, zjistí, co způsobilo výpadek, zda se sahalo na adresu, ke které neměl proces přístup, či se porušily práva pro zápis. Pokud se jednalo o legitimní přístup do paměti akorát na dané adrese ještě není načtena stránka je potřeba nalézt volný rámec a přiřadit ho tabulky stránek a zavolat navrácení z přerušení. Tento případ zároveň představuje koncept stránkování na žádost, kdy se může proces pokusit o přístup na validní adresu, ale stránka není zrovna načtená v paměti, to způsobí výpadek stránky, který bude obsluhovat právě funkce napsaná uživatelem. Protože se stránka načte až když je potřeba se tomuto říká stránkování na žádost nebo líné načítání (lazy loading).

4.3.4 Sdílení paměti

Tento případ se zabývá tím, jak sdílet paměť mezi více procesy. U tohoto případu jsem se inspiroval funkcemi pro sdílení paměti běžně používanými na linuxu a to operacemi **shmget** a **shmattach.** A právě implementaci těchto dvou funkcí ve zjednodušené podobě se požaduje po uživateli u tohoto případu.

První funkce by měla vytvořit strukturu reprezentující sdílenou část paměti a získat pro ni volné rámce. Tuto strukturu by pak měla přidat do seznamu se sdílenými částmi paměti.

Druhá funkce by měla s pomocí id vygenerovaného první funkcí nalézt vytvořený sdílený segment paměti a poté namapovat rámce v tomto segmentu do některých svých volných stránek. Tím, že si více procesů namapuje stejnou fyzickou část paměti do své virtuální paměti je vytvořena sdílená paměť, přes kterou mohou procesy komunikovat.



4.1: Rozložení webové stránky

4.3.5 Textový formulář

Jako komponentu pro vstup kódu od uživatele jsem použil existující editor kódu Ace[1] napsaný v JavaScriptu. Tento nástroj má spoustu zabudovaných funkcí, které usnadní uživateli psaní kódu, jako je například zvýrazňování syntaxe a automatické odsazování. Při načtení webové aplikace jsou všechny editory nastavené pro jazyk C.

Tento editor umožňuje velké množství konfigurace a nastavení. Tohoto je využito pro zobrazení chyb překladu, kdy se všechny řádky, na kterých našel překladač chybu, zvýrazní červenou barvou.

[1] https://ace.c9.io/

4.3.6 Komunikace se serverem

Pro komunikaci se serverem je využito WebSocketů. Pro každé odeslání kódu se naváže nové spojení, které je udrženo do doby, než server zpracuje požadavek a odešle výsledek. Prostřednictvím otevřeného spojení přes WebSocket se odesílají data ve formátu JSON. Všechna data dodržují stejný formát, odesílá se, jaká akce se má udělat, jaký případ se zrovna řeší a kód z editoru. Adresa a port serveru je definován v souboru index.html. Příklad, jaká data se zasílají, je demonstrováno na obrázku 4.2

```
{
   "case":"get_address",
   "action":"compile",
   "data":"void func(int arg){}"
}
```

4.2: Ukázka JSON protokolu

4.3.7 Zobrazení výsledků

Po odeslání dat serveru přijdou výsledky zpracování. Pokud operace byla kompilace, potom přijde informace o tom, zda byl překlad úspěšný či neúspěšný. U neúspěšného překladu se posílají také informace o chybách, které chybu překladu způsobily. Webová aplikace tyto informace ze serveru zpracuje, a pokud byl překlad úspěšný, tak zobrazí pod editorem zprávu zabarvenou zeleně, že překlad proběhl úspěšně. Jestli byl překlad neúspěšný, vypíšou se pod editor červeně chyby a případně se i označí chybné řádky v editoru, pokud je to možné.

Výsledky testování jsou ve formě popisu testu a informace, zda uspěl či neuspěl. Webová stránka projde všemi testy a přidá jejich popis do seznamu výsledků, zároveň jim i přidá správnou třídu, aby zobrazily příslušnou barvu. Všechny výsledky se poté vypíšou pod editorem. Uživatel uvidí jak zelené zprávy o testech, které se zdařily, tak i červené zprávy, které poukazují na test, který neprošel.

4.3.8 Vizualizace

Každý případ obsahuje ilustrační obrázek tabulky stránek, nebo nějakého jiného, k případu relevantního, obrázku. Tento obrázek je ve formátu SVG, obrázek byl vytvořen v programu Microsoft Visio[1], který umožňuje export do SVG. Výsledný soubor má formát XML kde jednotlivé elementy popisují všechny čáry, obrazce a další elementy obrázku. Zároveň dostane každý tento element unikátní ID. Toto XML je přidáno přímo do HTML souboru a prohlížeč je zobrazí jako obrázek. To že má každý element obrázku své ID umožňuje

[1] https://products.office.com/cs-cz/visio/flowchart-software

manipulaci s ním za pomocí JavaScriptu. Stačí najít element a upravit jeho styl či obsah, takto se dá měnit například barva či text.

Tohoto se u některých případů využívá pro vizualizaci toho, co daná funkce dělá a jak funguje. V průběhu funkce se dá narazit na stavy, každý tento stav se dá zachytit nějakou podobou obrázku a mezi těmito stavy se dá přepínat. Například pro získání fyzické adresy z virtuální za pomoci tabulky stránek se virtuální adresa rozdělí do tří částí a pak se jednotlivé části používají jako indexy do adresáře stránek, tabulky stránek, či ofset. Toto jsou jednotlivé fáze hledání fyzické adresy, k tomu lze udělat korespondující vizualizaci. Například když se zrovna hledá v adresáři stránek, tak se může zvýraznit část adresy reprezentující index do adresáře stránek a hledaný záznam v tabulce stránek.

Vytvořením stavového stroje v JavaScriptu, který v každém stavu manipuluje elementy SVG obrázku do požadovaného vzhledu, je možné dosáhnout zobrazení jednotlivých stavů. Tento stavový stroj poté měním v pravidelném intervalu a dosáhnu tak animace, která zobrazuje to, co by daný kód měl dělat.

4.4 Implementace serveru

Pro implementaci serveru jsem zvolil programovací jazyk Python, který má tu výhodu, že je v základu na většině distribucí Linuxu. Také umožňuje jednoduše využívat příkazy shellu. Funguje jako sprostředkovatel překladače GCC[1] pro webovou aplikaci a řeší překládání a spouštění testů a zasílání výstupů webové aplikaci. Spouští se následovně:

```
python3 server.py -s port --lxd LXD
```

Parametry **-s** a port jsou nepovinné. Parametr port specifikuje, na, kterém portu má server naslouchat. Defaultní hodnota je číslo portu 8765. Parametr **-s** či **--silent** způsobí, že server nebude vypisovat žádné informace. Přepínač --lxd slouží pro zapnutí testování v kontejneru a specifikuje jaký kontejner se má využít. Server totiž v základu vypisuje informace, co právě provádí, že přišlo nové připojení, jaká data přišla, jaká se odeslala a další.

4.4.1 Komunikace

Pro komunikaci se využívá WebSocketů. Server naslouchá na specifikovaném portu na příchozí spojení. Pro každé spojení potom spustí obslužnou funkci, která na základě přijatých dat začne vykonávat požadovanou akci a poté zašle aktivnímu spojení výsledná data. Pokud přijde nové spojení během obsluhy jiného, toto nové spojení je uloženo do fronty a začne se

[1] https://gcc.gnu.org/

obsluhovat ihned po skončení obsluhy předchozího spojení. Díky tomuto je server schopen obsluhovat více webových aplikací zároveň.

4.4.2 Překlad kódu

Pokud webová aplikace v datech požaduje překlad, tak server vezme kód, který přišel, a přiloží k němu knihovny tak, že přidá na začátek kódu kód pro zahrnutí hlavičkových souborů knihoven. Poté přidá ke kódu prázdnou **main** funkci jen proto, aby bylo možné kód přeložit. Kód poté uloží do souboru, jako název souboru je použit klíč WebSocketového spojení, díky tomu bude mít každé připojení jiný název souboru a nehrozí tak, že by si dvě spojení sahala na soubory. Tento soubor se poté přeloží, pro překlad je vytvořen Makefile soubor, který se využije pomocí **make**[1]. Server zavolá pravidlo, které pokud je třeba přeloží všechny knihovny a poté i soubor s kódem, ke kterému jsou tyto knihovny přiloženy. Aby **make** věděl, jaký soubor má přeložit, předává se mu argument SOURCE_NAME.

Server vezme výstup této operace a zkontroluje, zda operace skončila v pořádku, či nikoliv. Podle toho odešle webové aplikaci data. Pokud se operace zdařila a kód byl úspěšně přeložen, odešle webové aplikaci pouze informaci o úspěšném přeložení, jak ukazuje obrázek 4.2. Pokud se operace nezdařila a kód nebyl úspěšně přeložen, tak server zpracuje všechny chyby, které vyprodukoval překladač. Server tyto chyby zjednoduší a také upraví čísla řádků chyb tak, aby korespondovala s čísly řádků na webové aplikaci. Každá tato chyba se pak přidá do pole v odesílaných datech a pošle se webové aplikaci, jak je zobrazeno na obrázku 4.4.

Po odeslání dat si server po sobě uklidí a vymaže jak vytvořený soubor se zdrojovým kódem, tak i případný vygenerovaný binární soubor.

```
{
   "action":"compile",
   "case":"get_address",
   "result":"success"
}
```

4.3: Ukázka JSON při úspěchu překladu

[1] https://www.gnu.org/software/make/manual/make.html

```
{
   "action":"compile",
   "case":"get_address",
   "result":"failure",
   "output":
      [
         "5:5: error: 'abc' undeclared"
         "9:1: error: expected ';' before '}' token"
      ]
}
```

4.4: Ukázka kódu JSON při chybě překladu

4.4.3 Testování kódu

Postup obsluhy, když webová aplikace požaduje otestování kódu, je obdobný jako u překladu kódu. Kód se vezme, přidají se k němu knihovny, navíc se k němu ale přidá i testovací kód. Server podle toho, jaký případ se právě řeší, vybere soubor se správným testovacím kódem a přidá jeho obsah na konec přijatého kódu.

Po překladu za pomocí **make** se nekouká na výstup překladu, ale místo toho je výsledný binární program spuštěn a server vezme jeho výstup, zpracuje ho a odešle webové aplikaci. Výstup by měl obsahovat výstupy jednotlivých testů, ale pokud kód není správný, může se něco pokazit, program spadnout a testy vůbec neproběhnout. I tyto případy server ošetřuje a zašle informace o tom, že například kód vyústil v SEGMENTATION_FAULT webové aplikaci.

Stejně jako při kompilaci, tak i po testování si po sobě server uklidí všechny vytvořené soubory.

4.4.4 Bezpečné testování kódu

Server umožňuje testovat vygenerovaný kód na linuxovém kontejneru. Tato funkcionalita se spustí pomocí přepínače --lxd LXD, kde LXD specifikuje vytvořený kontejner. Server podporuje pouze kontejnery LXD[1]. Aby mohl server tento kontejner využívat, potřebuje, aby již byl kontejner vytvořený a taky aby měl práva k němu přistoupit což znamená pustit server se správcovskými práci. Nebo přidat uživateli, který server spouští právo na používání lxd pomocí příkazu **newgrp lxd**.

Jakmile jsou tyto podmínky splněny, server bude vygenerované binární soubory z uživatelského kódu kopírovat do kontejneru, tam je spustí a vezme jejich výstup po tom daný soubor smaže.

Díky tomu, že se bude kód pouštět v kontejneru v izolovaném prostředí není možné, aby vykonávaný kód, jakkoliv destabilizovat či poškodil hostitelský systém.

[1] https://linuxcontainers.org/lxd/introduction/

4.4.5 Bezpečnostní kontrola nepovolených funkcí

Protože může přijít libovolný kód z webové aplikace, jsou z důvodu bezpečnosti podniknuté kroky, které by měly omezit, co může kód vykonávat. Kód se za pomocí regulárních výrazů projde a pokusí se vyhledat nepovolené funkce. Pokud nějaké najde, odešle pro každou z nich chybové hlášení webové aplikaci, která to zobrazí serveru. Názvy nepovolených funkcí má server uloženy v seznamu. Funkce byly vybrány tak, že jsem si prošel všechny funkce ze standartní knihovny **stdlib.h** a vybral všechny, které buď alokovaly paměť (malloc, calloc, realloc) nebo mohly ovlivnit prostředí (system).

4.5 Implementace knihoven

Aby uživatel nemusel psát celý paměťový systém a mohl se soustředit pouze na svoji funkci, bylo zapotřebí, aby byly vytvořené funkce a datové struktury, které by mohl využívat, a na kterých by mohl stavět. Jelikož uživatel programuje funkce v jazyce C, tak i všechny knihovny jsou implementovány v jazyce C.

Knihovny jsou implementované jako moduly, což znamená, že části paměťového systému jsou implementovány ve zdrojovém **.c** souboru a k němu je vytvořen hlavičkový **.h** soubor. Všechny funkce, datové struktury a výčtové typy, které mají být přístupné pro použití uživatele, jsou v hlavičkovém souboru. Celkem bylo vytvořeno pět modulů, kde každý implementuje část funkcionality.

Modul s názvem PageTableEntry obsahuje datový typ reprezentující záznam v tabulce stránek a funkce pro manipulaci s ní, jako například: (nastavení adresy, získání adresy, nastavení atributu). Také obsahuje výčtový typ s hodnotami reprezentující atributy, které jsou použitelné ve jmenovaných funkcích.

Modul PageDirectoryEntry je obdoba PageTableEntry, takže také obsahuje datový typ reprezentující záznam v adresáři stránek, funkce pro její manipulaci a výčtový typ reprezentující atributy.

Modul TLB obsahuje datový typ pro tabulku a funkce pro její manipulaci jako vyhledání záznamu, přidání a smazání záznamu.

Modul SHM se zabývá sdílením paměti. Obsahuje datové struktury, které reprezentují sdílené segmenty paměti a informace o nich. Také obsahuje funkce pro práci se seznamem těchto struktur jako je přidání záznamu, získání záznamu podle id či nalezení id záznamu podle klíče.

Modul MMU je nejvyšší modul, který využívá všech předešlých modulů. Reprezentuje jednotku pro správu paměti a její funkcionalitu. Obsahuje pole záznamů, které reprezentuje adresář stránek, a právě přes toto pole bude uživatel přistupovat k tabulkám stránek a s tímto polem bude pracovat v každém případu, který bude řešit. Krom toho obsahuje také funkce, které jsou potřebné pro vyřešení případů, některé funkce fungují pouze jako

abstrakce nějaké funkcionality paměťového systému a pouze vracejí nějakou konstantu, což ale uživatel nepozná a testy toto očekávají. Tento modul také obsahuje funkce, které mohou využívat testy, jsou to především různé inicializační funkce, které zaplní adresář stránek a tabulky stránek nějakými daty, aby testy mohly testovat funkcionalitu na nějakých datech. Také jsou tu implementovány funkce ze všech případů pouze pod změněným názvem, které slouží zejména pro testování, protože některé testy jich využívají pro porovnání výstupu funkce napsané uživatelem a již hotové funkce. Tyto funkce nejsou nikde popsané, takže uživatel je nebude moci používat.

Všechny tyto moduly se nachází ve složce s názvem **include**, která je umístěna ve složce serveru. Server tyto moduly podle potřeby přeloží na objektové soubory a ty poté přidá k uživatelskému kódu pro překlad.

4.6 Implementace testů

Testy slouží pro otestování funkcionality kódu napsaného uživatelem. Jsou napsány v jazyce C a pro každý případ existuje jeden soubor s testy, obsahuje funkci **main**, která provádí inicializaci a pak volá jednotlivé testy, které testují funkcionalitu funkce z daného případu.

Výstupem každého testu je jednoduchý text skládající se ze dvou částí, první část říká, jestli test proběhl úspěšně nebo neúspěšně a druhá část obsahuje popis. Tento popis buď obsahuje gratulaci uživateli, že jeho kód správně provedl nějaký úkol nebo naopak obsahuje upozornění, že si jeho kód neporadil s nějakým úkolem.

Testů v testovacím souboru může být libovolné množství, vytvořil jsem takové množství testů, aby pokryly všechny cesty v kódu testované funkce. Testy mohou testovat různé věci v závislosti na tom, jaká funkce je testována, ale často se testují návratové hodnoty, či se kontrolují po vykonání funkce datové struktury, kde se kontroluje, zda funkce vytvořila, upravila a namapovala stránky tak, jak měla. Některé testy využívají již vytvořené referenční funkce, kterou jsem pečlivě testoval a porovnával ji s funkcí od uživatele a test pak porovnává, jestli dělají to stejné.

Výstup všech testů poté vezme server a odešle webové aplikaci, která výsledky zobrazí uživateli, aby věděl, co udělal dobře, či naopak co by měl ještě opravit.

4.7 Nasazení a požadavky systému

Pro nasazení systému je zapotřebí nejprve zprovoznit server, poté nakonfigurovat webovou aplikaci tak, aby použila IP adresu a port serveru. Server je možno spustit jednoduchým příkazem takto:

```
python3 server.py
```

Jakmile server běží, stačí u webové aplikace v souboru index.html na začátku JavaScriptového kódu upravit proměnné ip_address a port s IP adresou počítače, na kterém běží server a na jakém portu naslouchá.

Pak už je možno využívat webové aplikace jednoduše tak, že se spustí index.html v prohlížeči. Webová aplikace má všechny nástroje, knihovny a závislosti přibalené s sebou, takže nepotřebuje připojení k internetu a může fungovat offline, pokud je server umístěn ve stejné lokální síti jako počítač využívající webovou aplikaci.

4.7.1 Minimální požadavky

Jelikož server využívá async a await, což jsou poměrně nové funkce, v pythonu je pro spuštění serveru požadovaná verze pythonu 3.5 a také je zapotřebí mít přes pip nainstalovaný balíček websockets[1].

Kód testů a knihoven, testy a uživatelský kód využívají standardu C11 jazyka C. Pro překlad je proto potřeba verze GCC, která tento standard podporuje, což je jakákoliv verze od 4.7 a výše.

Kvůli využití některých moderních technologií u webové aplikace, je doporučeno využívat aktuální verzi prohlížeče, na kterém je jisté, že vše pojede bez problémů. V případě použití Internet Exploreru je zapotřebí minimálně verze 9.

4.8 Testování systému

Pro systém byla vytvořena testovací sada, která slouží pro ověření funkčnosti webové aplikace serveru. Testy byly vytvořeny pomocí nástroje SnapTest[2], což je doplněk do internetového prohlížeče Chrome, který umožňuje jednoduchou tvorbu a správu testů postavených na Selenium.

Nástroj funguje tak, že umožňuje nahrávat uživatelské akce ve webovém prohlížeči, tyto akce se krok po kroku nahrávají a mezi tyto kroky se dají přidávat i asserty, kterými lze ověřit, zda webová aplikace reagovala na akci tak, jak měla.

Vytvořené testy pro tuto práci kontrolují mimo jiné, navigační menu, že každý odkaz vede na správnou stránku a u každého případu zkouší odeslání příkazu ke kompilaci, jednou se správným kódem a jednou se špatným kódem a kontroluje se zda, server pošle nazpět správná data a zda je webová aplikace zobrazí, jak by měla. Dále se také testuje použití kvůli bezpečnosti zakázaných funkcí a jestli webová aplikace správně zobrazí chybové hlášení o použití zakázaných funkcí.

Tento doplněk umožňuje i pouštění testů, ale bohužel jelikož je pro používaní nutný účet, tak by nikdo kromě mě nemohl testy používat, protože vlastním používaný účet.

[1] https://pypi.python.org/pypi/websockets
[2] https://www.snaptest.io/

Naštěstí umí tento doplněk i vygenerovat testovací kód z vytvořených testů. Tyto vygenerované testy lze najít ve adresáři webové aplikace ve složce snaptests.

Pro spuštění těchto testů je potřeba vyřešit několik závislostí, jelikož ty testy používají testovací framework zvaný nightwatch, bez kterého testy nelze spustit. Závislosti pro testování jsou uvedené v souboru se závislostmi, takže stačí tyto balíčky nainstalovat pomocí **`npm install`**. Pak již lze spustit testy za pomocí příkazu **`node_modules/.bin/nightwatch -conf nightwatch.conf.js`** a počkat na jejich provedení a výsledky.

Kapitola 5

Závěr

V této práci jsou položeny teoretické základy potřebné pro implementaci této diplomové práce. Tyto teoretické základy jsem pečlivě nastudoval a pochopil jsem paměťový systém a jednotku správy paměti dostatečně tak, abych byl schopen vytvořit jeho simulování a vizualizaci. Krom úvodu do paměťového systému jsou tu i detailně popsané webové technologie, které byly využity nebo jakkoliv ovlivnili vývoj této práce.

Krom toho práce obsahuje i představení a návrh aplikace. Navrhnul jsem simulátor jako webovou aplikaci. Rozhodl jsem se aplikaci rozdělit na několik částí, u jednotlivých částí je popsáno, co budou mít za funkce a zodpovědnosti a jak je budou provádět.

Je tu popsáno, i jak byla celá práce naimplementována, jak jsem vytvořil jednotlivé části, a proč jsem je vytvořil takové jaké jsou. Implementace části je do detailu popsaná, tak aby po přečtení měl čtenář velice dobrou představu, co vše vytvořený systém umí a jak funguje.

Pro tuto práci jsem vytvořil testovací sadu, která testuje základní funkcionalitu a bezpečnost. Je tu popsané, jak byly testy vytvořeny i co dělají. A jaké možnosti aplikace nabízí z hlediska bezpečnosti.

Tato práce má stále mnoho míst, kam růst. Má mnoho míst, kde by se dala rozšířit či vylepšit. Jako zřejmé rozšíření se nabízí vytvoření více případů, paměťový systém je rozsáhlý a obsahuje mnoho konceptů, které by se daly vytvořit jako případ pro uživatele na vytvoření. Nabízela by se například manipulace s TLB, pro kterou je vytvořena knihovna, která ale nakonec nebyla využita. Další možností rozšíření by bylo použít místo tabulky stránek invertovanou tabulku stránek. Dále by také bylo možno vyzkoušet vytvořit případy pro jinou architekturu, tato práce se zaměřuje pouze na 32 bitovou architekturu, přidáním případů pro 64 bitovou architekturu by bylo možné vidět rozdíly mezi nimi.

Výsledkem této práce je systém, který prostřednictvím případů představuje koncepty okolo stránkování paměti. Každý případ představí uživateli koncept a dá mu možnost si vyzkoušet tento koncept naprogramovat. Na svůj kód dostane od aplikace odezvu a koncepty mu navíc budou vizualizované, což pomůže s pochopením daného konceptu.

Věřím, že výsledný systém by mohl pomoci s výukou konceptů z paměťového systému. Sloužit by mohl především jako sekundární nástroj pro učení, kdy by pomohl prohloubit znalosti o věcech, o kterých uživatel už něco ví, ale pořád o nich nemá hluboké znalosti.

Citovaná literatura

[1] A. S. Tanenbaum a A. Woodhull, Operating systems Design and Implementation, 3rd edition, Amherst: Prentice Hall, 2006.

[2] University of Nottingham, „Relocation and Protection," [Online]. Available: http://www.cs.nott.ac.uk/~pszgxk/courses/g53ops/Memory%20Management/MM04-relocation.html. [Přístup získán 4. 1. 2017].

[3] B. Jacob a T. Mudge, „Virtual memory: issues of implementation," *Computer,* sv. 31, pp. 33-43, 1998.

[4] P. J. Denning, „Virtual memory," *ACM Computing Surveys (CSUR),* sv. 2, č. 3, pp. 153-189, 1970.

[5] Wikipedie, „Page table," [Online]. Available: https://en.wikipedia.org/wiki/Page_table. [Přístup získán 4. 1. 2017].

[6] A. Silberschatz, P. B. Galvin a G. Gagne, Operating System Concepts, 9. editor, 2013.

[7] Intel, „Intel® 64 and IA-32 Architectures Software Developer Manual: Vol 3," [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-system-programming-manual-325384.pdf. [Přístup získán 5. 1. 2017].

[8] wiki.osdev.org, „Paging," [Online]. Available: http://wiki.osdev.org/Paging. [Přístup získán 6. 1. 2017].

[9] T. L. D. Project, „Shared Memory," 29 Březen 1996. [Online]. Available: http://www.tldp.org/LDP/lpg/node65.html.

[10] Linux Kernel Organization, „Page Table Management," [Online]. Available: https://www.kernel.org/doc/gorman/html/understand/understand006.html. [Přístup získán 9. 1. 2017].

[11] TutorialsPoint, „Operating System - Virtual Memory," [Online]. Available: https://www.tutorialspoint.com/operating_system/os_virtual_memory.htm. [Přístup získán 9. 1. 2017].

[12] A. Silberschatz, P. B. Galvin a G. Gagne, Operating Systems Concepts, 2012.

[13] A. S. Tanenbaum, „Page Replacement Algorithms," [Online]. Available: http://www.informit.com/articles/article.aspx?p=25260&seqNum=5. [Přístup získán 9. 1. 2017].

[14] A. M. McHoes a I. M. Flynn, Understanding Operating Systems, 6. editor, 2011.

[15] J. Duckett, HTML and CSS: Design and Build Websites, John Wiley & Sons, Inc., 2011.

[16] J. Duckett, JavaScript and JQuery: Interactive Front-End Web Development, John Wiley & Sons, Inc., 2014.

[17] J. Tesařík, L. Doležal a C. Kollmann, „User interface design practices in simple single page web applications," v *2008 First International Conference on the Applications of Digital Information and Web Technologies (ICADIWT)*, Ostrava, 2008.

[18] Microsoft, „ASP.NET - Single-Page Applications: Build Modern, Responsive Web Apps with ASP.NET," [Online]. Available: https://msdn.microsoft.com/en-us/magazine/dn463786.aspx?f=255&MSPPError=-2147217396. [Přístup získán 9. 1. 2017].

[19] Facebook, „React," [Online]. Available: https://facebook.github.io/react/. [Přístup získán 9. 1. 2017].

[20] P. Pololáník, Fotosoutěž na Facebook, Brno, 2014/2015.

[21] B. Lin, Y. Chen, X. Chen a Y. Yu, „Comparison between JSON and XML in Applications Based on AJAX," v *2012 International Conference on Computer Science and Service System*, Nanjing, 2012.

[22] Mozilla, „WebSockets," [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API. [Přístup získán 10. 1. 2017].

[23] A. Quint, „Scalable vector graphics," *IEEE MultiMedia,* sv. 10, č. 3, pp. 99-102, 4 Srpen 2003.

[24] C. Saternos, Bootstrap: Responsive Web Development, O'Reilly Media, 2013.

[25] M. Cantelon, N. R. a M. Harter, Node.js in Action., Manning Publications, 2014.

[26] Red Hat, „What's a Linux container?," [Online]. Available: https://www.redhat.com/en/containers/whats-a-linux-container.

Příloha A

Obsah CD

- /text/ - technická zpráva
- /web/ - webová aplikace a zdrojové kódy
- /server/ - serverová aplikace a zdrojové kódy
- /images/ - obrázky vytvořené pro tuto práci