Katedra informatiky, FEI VŠB-TUO, Petr Olivka.

Tento text je neautorizovaný a nerecenzovaný překlad doporučené literatury:

"Andrew S. Tanenbaum, Operating Systems: Design and Implementation",

a je určen jen pro studijní účely.

3 Vstupy a výstupy

Jedna z hlavních funkcí operačního systému je řízení všech vstupní a výstupních zařízeních počítače. Musí vysílat příkazy do zařízení, zachytávat přerušení a reagovat na chyby. Taky se stará o rozhraní mezi zařízeními a zbytkem systému, který je jednoduchý a snadný pro používání. No a nakonec by rozhraní mělo být stejné pro všechna zařízení (device independence). V/V kód představuje významnou část z celého operačního systému. Jak spravuje operační systém V/V je předmětem této kapitoly.

Osnova kapitoly je následující. Nejprve se podíváme okrajově na na principy V/V hardware a pak probereme V/V software obecně. V/V software může být rozčleněn do vrstev, kde každá vrstva má svou jasně definovanou činnost. Podíváme se na tyto jednotlivé vrstvy, co dělají a jak na sebe navazují.

Pak přijde část o zablokování. Budeme přesně definovat zablokování, ukážeme, jak k němu dojde, představíme dva modely pro jeho analýzu a budeme se bavit o algoritmech pro prevenci jejich vzniku.

3.1 Principy V/V hardware

Různí lidé mají na V/V hardware rozdílný pohled. Inženýr elektrotechnik se na ně dívá jako na čipy, dráty, napájení, motor a všechny další komponenty tvořící hardware. Programátory zajímá rozhraní pro programování – příkazy které zařízení přijímá, funkce které vykonává a chyby, které může vrátit zpět. Zde se budeme zabývat programováním V/V zařízení, ne jejich návrhem, výrobou a údržbou a náš zájem se omezí na to, jak se hardware programuje, ne na jeho vnitřní činnost. Přesto programování mnoha V/V zařízení je úzce spojeno s jich vnitřními operacemi. V následujících třech kapitolách se podíváme na obecný základ souvislostí V/V hardware a programování.

3.1.1 V/V zařízení

V/V zařízení se dělí zhruba na dvě kategorie: **bloková zařízení** (block device) a **znaková zařízení** (character device). Blokové zařízení je to, které ukládá informace v blocích stejné velikosti a každý blok má vlastní adresu. Obvykle je velikost bloku od 512 do 32768 bytů. Základní vlastností blo-

kových zařízení je schopnost číst a zapisovat každý blok nezávisle na všech ostatních. Nejobvyklejším blokovým zařízením je disk.

Pokud se podíváme podrobněji, tak hranice mezi zařízením, které je a není adresováno blokově, není přesně definována. Každý bude souhlasit, že disk je blokově adresovatelné zařízení, protože bez ohledu na pozici hlaviček můžeme kdykoliv přejít na jiný cylindr a pak počkat na požadovaný blok natočením pod čtecí hlavičkou. Teď si vezměme 8 mm páskovou DAT mechaniku pro zálohování disku. Pásky obecně obsahují bloky stejné velikosti. Pokud je mechanice dán příkaz ke čtení bloku N, tak může vždy převinout pásku a postupovat vpřed, dokud nenarazí na blok N. Je to operace analogická s přechodem na požadovaný sektor na disku, jen trvající podstatně déle. I když by byla možnost použít pásku jako blokové zařízení s náhodným přístupem, je to poněkud zdlouhavé a obvykle se tedy takto nepoužívá.

Jiným typem V/V zařízení je znakové. Znakové zařízení předává nebo přijímá proud znaků bez jakékoliv blokové struktury. Není adresovatelné a nezná operaci vyhledávání (seek). Tiskárna, síť, myš, sériový port a mnoho dalších zařízení, která nejsou podobná diskům a můžeme se na ně dívat jako na znaková zařízení.

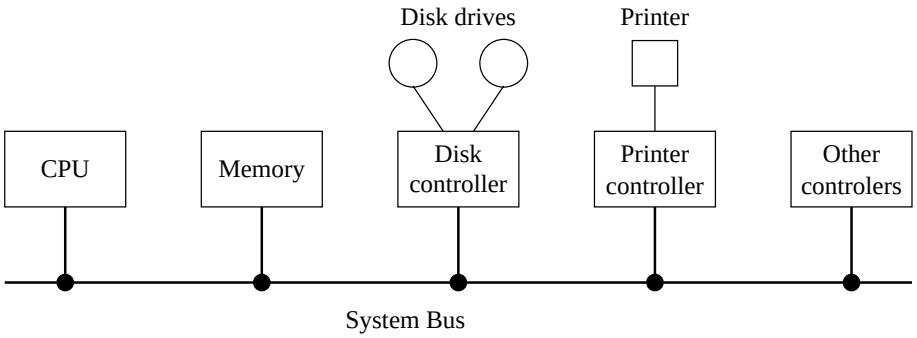
Toto schéma dělení není nijak dokonalé. Mnoho zařízení tomu neodpovídá. Například hodiny, nejdou adresovatelné blokově. Nemohou ani generovat nebo přijímat proud dat. Vše co způsobují, je přerušení v definovaném intervalu. Paměťově mapovaná obrazovka tomuto modelu taky nevyhovuje. Stále je ale model znakových a blokových zařízení nejobecnější, který se používá jako základ pro tvorbu software operačního systému komunikujícího s V/V zařízeními. Například souborový systém komunikuje s abstraktním blokovým zařízením a nechává část závislou na zařízení nižší úrovni software, nazývanou **ovladač zařízení** (device driver).

3.1.2 Řídící jednotka

V/V jednotka je obvykle složena z mechanické a elektrické části. Obě části se od sebe dají většinou oddělit a umožňují tak určitou modularitu. Elektronickou část nazýváme **řídící jednotkou** (device controller), nebo **adaptérem**. V personálních počítačích je jako karta s tištěnými spoji vkládána do slotu sběrnice na základní desce. Mechanickou částí rozumíme zařízení samotné.

Řídící karta má na sobě obvykle konektor pro připojení zařízení kabelem. Většina řadičů může ovládat dvě, čtyři, nebo i více stejných zařízení. Pokud je rozhraní mezi řadičem a zařízením standardní, jako třeba ANSI, IEEE nebo ISO, mohou pro ně výrobci vyrábět řadiče nebo zařízení. Například pro IDE nebo SCSI rozhraní vyrábí zařízení celá řada výrobců.

Rozdíl mezi zařízením a řadičem zmiňujeme záměrně, protože operační systém komunikuje téměř vždy s řadičem, ne se zařízením. Většina malých počítačů používá jednu sběrnici, jako na obrázku 1, pro komunikaci mezi CPU a řadičem. Velké sálové počítače používají odlišný model s více sběr-



Obrázek 1: Typický model propojení CPU, paměti a řadičů.

nicemi a specializovanými V/V procesory, nazývanými **V/V kanály** (I/O channels), částečně přebírající zátěž hlavního procesoru.

Rozhraní mezi řadičem a zařízením je často velmi nízko-úrovňové. Například disk se může formátovat s 16 sektory po 512 bytech na stopu. To co ale ze zařízení vystupuje, je obvykle proud bitů, začínající **hlavičkou** (preamble), pak 4096 bitů ze sektoru a nakonec kontrolní součet, někdy nazývaný **ECC** (Error–Correcting Code). Hlavička se zapisuje při formátování a obsahuje číslo cylindru a sektoru a další synchronizační údaje. Úkolem řadiče je konvertovat proud bitů na blok dat a provést kontrolu a případně opravu chyb. Blok bytů je nejprve poskládán bit za bitem do bufferu uvnitř řadiče. Po zkontrolování kontrolního součtu je blok prohlášen za bezchybný a může se zkopírovat do hlavní paměti.

Řadič pro CRT monitor taky funguje na nejnižší úrovni jako sériové bitové zařízení. Čte data obsahující znaky a generuje signály pro modulaci řízení paprsku CRT monitoru, aby se zobrazily na obrazovce. řadič také generuje signály pro horizontální a vertikální synchronizaci paprsku monitoru. Kdyby tu nebyl řadič pro CRT monitor, programátor operačního systému by musel programovat analogové řízení monitoru. S řadičem jen operační systém inicializuje několik parametrů, jako je počet znaků na řádku a počet řádků na obrazovce a starost o řízení paprsku přenecháme řadiči.

Každý řadič má několik registrů pro komunikaci s CPU. Na některých počítačích jsou registry jako součást paměťového adresního prostoru. Toto schéma se nazývá **paměťově mapované V/V**. Například Motorola 680x0 používá tento princip. Jiné počítače mají speciální adresní prostor pro V/V, kde má každý řadič vyhrazen svůj prostor. Adresa se přiřazuje dekódovací logikou spojenou s řadičem. Někteří výrobci zařízení kompatibilních s IBM-PC používají jiné adresy než jaké původně IBM. Kromě V/V bran používá mnoho řadičů i přerušení, aby bylo možno CPU říct, že jsou jejich registry připraveny pro čtení či zápis. Přerušení je v první řadě určitá elektrická událost. Žádost o přerušení (IRQ - Interrupt ReQuest) je fyzický vstup do

řadiče přerušení. Počet takových vstupů je omezený. Počítače IBM-PC mají jen 15 IRQ pro zařízení. Některé řadiče jsou integrovány přímo na základní desce počítače, jako například ovladač klávesnice v IBM PC. V případě desek vkládaných do slotů sběrnice na základní desce se příslušná konfigurace zadává přepínači nebo propojkami, aby nedocházelo ke konfliktům a novější systémy využívají systém Plug and Play, kdy se IRQ nastavuje programově. V tabulce 1 je příklad některých zařízení v počítači IBM PC, jejich adres V/V bran, přerušení a odpovídajících vektorů přerušení.

Tabulka 1: Příklad řadičů, jejich adres a IRQ typický pro IBM-PC.

V/V řadič	**V/V adresa**	**IRQ**	**Int. vector**
Časovač	0x040-0x043	0	8
Klávesnice	0x060-0x063	1	9
Pevný disk	0x1F0-0x1F7	14	118
COM2	0x2F8-0x2FF	3	11
LPT1	0x378-0x37F	7	15
Disketa	0x3F0-0x3F7	6	14
COM1	0x3F8-0x3FF	4	12

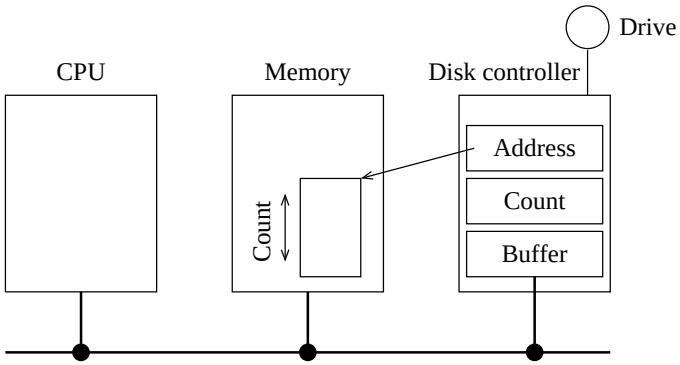
Operační systém provádí V/V operace posíláním příkazů řadiči. Většina příkazů má parametry, které se zapisují do registrů řadiče. Když je příkaz přijat, nechá CPU řadič pracovat samostatně a věnuje se jiné práci. Když je příkaz proveden, vyvolá řadič přerušení, aby operační systém získal CPU a mohl vyhodnotit výsledek operace. CPU přečte výsledek a status, nebo další informační byty z registrů řadiče.

3.1.3 Přímý přístup do paměti (DMA - Direct Memory Access)

Mnoho řadičů, typicky pro bloková zařízení, využívají **DMA**. Abychom si vysvětlili princip DMA, podívejme se nejdříve, jak probíhá čtení z disku bez DMA. Nejprve přečte řadič blok z disku sériově, bit za bitem, dokud není blok celý, do vnitřního bufferu. Dále pak ověří kontrolní součet, aby se vyloučily případné chyby čtení. Pak řadič vyvolá přerušení. Když se spustí obsluha přerušení, přečte se blok disku z řadiče po jednotlivých bytech ve smyčce, kdy se každý byte nejprve načte do registru a pak se uloží do paměti.

Samozřejmě, programová smyčka čtoucí jednotlivé byty z řadiče plýtvá procesorovým časem. DMA bylo navrženo, aby zbavilo CPU těchto nízkoúrovňových činností. Při použití DMA předává CPU dvě informace, tedy kromě adresy bloku disku: adresu paměti a počet bytů pro přenos, jako na obrázku 2.

Jakmile řadič přečte celý blok z disku a ověří bezchybnost, zkopíruje první byte do hlavní paměti na adresu specifikovanou v DMA. Pak zvýší



Obrázek 2: DMA - přenos dat z interního bufferu do paměti bez účasti CPU.

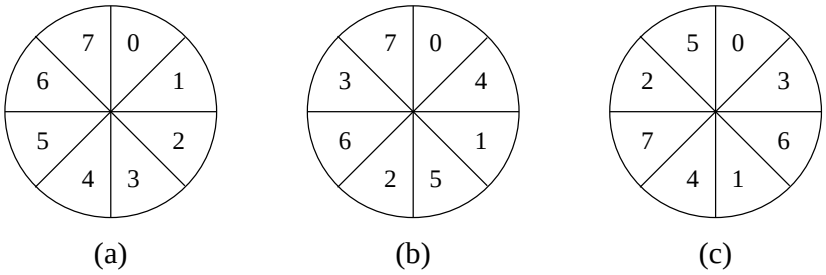
adresu DMA a sníží počet přenášených bytů. Tento proces se opakuje, dokud není počet bytů pro přenos nulový a pak řadič vyvolá přerušení. Když se spustí kód operačního systému, už se nemusí kopírovat data do paměti, už tam jsou.

Možná se ptáte, proč řadič neuloží data do paměti hned, jakmile je přečte z disku. Tedy proč vlastně potřebuje vnitřní buffer? Když začne přenos dat, tak bity přicházejí z disku konstantní rychlostí, ať už je řadič připravený, nebo ne. Kdyby se řadič pokoušel zapisovat data přímo do paměti, tak by musel jít s každým bytem přes systémovou sběrnici. Kdyby však byla sběrnice obsazena jiným zařízením, řadič by musel počkat. Pokud by tedy přišel další byte z disku a předchozí ještě nebyl uložen v paměti, musel by jej řadič někam odložit. Pokud by byla sběrnice příliš zatížená, řadič nakonec uloží jen několik bytů a přibude spousta další administrace kolem. Když je blok dat uložen do interního bufferu, není potřeba sběrnici, dokud nezačne DMA. A návrh tohoto řadiče je podstatně jednodušší, protože přenos dat do paměti není časově kritický.

Uvedený dvoustupňový princip bufferování má důležité důsledky pro V/V výkon. Zatím, co jsou data přenášena z řadiče do paměti, ať už pomocí CPU nebo řadičem, proběhne pod diskovou hlavičkou další sektor a přijdou bity do řadiče. Jednoduché řadiče nemohou zvládat vstup i výstup současně, tak po dobu přenosu dat do paměti je sektor probíhající pod hlavičkami ztracen.

Výsledkem je schopnost řadiče číst každý druhý blok. Přečtení celé stopy pak vyžaduje dvě celé otáčky, jednu pro liché a druhou pro sudé bloky. Pokud bude čas přenosu dat do paměti větší, než čtení jednoho bloku, bude nutné přečíst blok a dva nebo i více bloků přeskočit.

Přeskakování bloků, aby měl řadič čas na přenos dat do paměti, se nazývá **prokládání** (interleaving). Když se disk formátuje, bloky jsou číslovány v pořadí prokládacího faktoru. Na obrázku 3(a) vidíme disk s osmi bloky



Obrázek 3: (a) Bez prokládání. (b) Jednoduché prokládání. (c) Dvojité prokládání.

na stopu bez prokládání. Na obrázku 3(b) je jednoduché prokládání a na obrázku 3(c) prokládání dvojité.

Hlavním důvodem tohoto číslování bloků je dát možnost operačnímu systému číst bloky v sekvenčním pořadí a dosahovat maximální rychlosti, jaké je hardware schopen. Pokud jsou bloky číslovány jako na obrázku 3(a) a řadič může číst jen každý druhý blok, tak operační systém, který si alokoval 8-blokový soubor v sekvenčním pořadí bloků, bude potřebovat 8 otáček pro přečtení bloků 0 až 7 v požadovaném pořadí. Jistě, dá se to řešit i programově, ale je jednodušší naformátovat disk a přenechat to řadiči.

Ne všechny počítače používají DMA. Argumentem proti je fakt, že procesory jsou často daleko výkonnější než řadič a jsou schopny zvládnout tento úkol daleko rychleji. Pokud máme rychlý procesor a nemáme pro něj nic jiného na práci, než čekat na pomalejší DMA, pak není co řešit. Odstranění řadiče DMA a přenechání programového řešení jen procesoru také ušetří určité peníze.

3.2 Principy V/V software

Nechejme chvíli hardware stranou a podívejme se na software. Hlavní cíl V/V software je jasný. Základní myšlenka je navrhnout software jako řadu vrstev. Od nejnižší, která má za úkol eliminovat zvláštnosti hardware, až po nejvyšší, sloužící jako jednoduché, přehledné a regulérní rozhraní pro uživatele. V následující kapitole se podíváme, jak se dá tohoto cíle dosáhnout.

3.2.1 Cíle V/V software

Klíčovou záležitostí návrhu V/V software je **nezávislost na zařízení** (device independence). Znamená to, že je možné napsat program, který může číst soubory z diskety, pevného disku nebo CD-ROMu, bez nutnosti měnit program pro každý typ zařízení. Každý by měl mít možnost napsat příkaz:

```
sort < input > output
```

a měl by fungovat se vstupem z diskety, disku nebo klávesnice a výstup může být opět na disketu, disk, nebo na obrazovku. Je věcí operačního systému, aby eliminovat problémy, které plynou z odlišnosti jednotlivých zařízení, která potřebují zcela odlišný ovladač pro zápis na zařízení.

S nezávislostí na zařízení úzce souvisí i **jednotnost pojmenovávání** (uniform naming). Jménem souboru nebo zařízení může být řetězec nebo číslo, a to nezávisle na zařízení. V UNIXu mohou být všechny disky spojeny do libovolného jediného hierarchického systému a uživatel se nemusí zajímat, které jméno patří kterému zařízení. například disketa se může **montovat** do adresáře */usr/backup* a tak kopírování do adresáře */usr/backup/monday* bude kopírovat soubory na disketu. Takto jsou všechny soubory adresovány jednotně: „jménem cesty".

Další důležitou vlastností V/V software je ošetření chyb. Chyby se musí řešit tak blízko hardware, jak je to jen možné. Když řadič detekuje chybu čtení, měl by se ji sám pokusit opravit, pokud může. Pokud to nelze, měl by to řešit ovladač zařízení, pravděpodobně pokusem přečíst daný blok znovu. Mnoho chyb je jen dočasných, jako chyba čtení způsobená částečkami prachu na čtecí hlavě, která zmizí při opakovaní operace. Jen v případě, že nižší vrstvy nejsou schopny si s problémem poradit, měly by o tom informovat vrstvy vyšší. V mnoha případech se dají chyby řešit zcela korektně na nižší úrovni, aniž by vyšší vrstvy o těchto chybách něco věděly. Avšak jinou klíčovou vlastností jsou synchronní (blokující) kontra asynchronní (přerušením řízené) přenosy. Většina V/V zařízení je asynchronních – CPU spustí přenos a odejde dělat něco jiného, dokud nepřijde přerušení. Uživateli se ale daleko snadněji píšou programy, když jsou V/V operace blokující – zavoláním příkazu READ se program automaticky pozastaví, dokud nejdou data v bufferu. Je na operačním systému, aby z přerušením řízených operací vytvořil operace blokující pro program uživatele.

A posledním návrhem kterým se budeme zabývat, jsou zařízení sdílená a vyhrazená. Některá zařízení, jako třeba disk, může v jednu chvíli sdílet více uživatelů. Mít otevřeno současně několik souborů nemůže způsobovat problémy. Jiná zařízení, jako například pásky, jsou vyhrazena jednomu uživateli, dokud ten neukončí svou práci. Pak může mít páskovou mechaniku někdo jiný, Mít dva nebo více uživatelů zapisujících promíchaná data na jednu pásku, by bylo úplně k ničemu. Návrh vyhrazených zařízení ale přinesl různé problémy. A opět operační systém musí být schopen manipulovat se sdílenými i vyhrazenými prostředky tak, aby problémům předcházel.

Všechny cíle lze dosáhnout srozumitelnou a účelnou formou - strukturováním V/V software do čtyř vrstev:

1. ovladač přerušení,
2. ovladač zařízení,
3. část operačního systému nezávislá na zařízení,

4. uživatelské programy.

V následujících částech se podíváme na jednotlivé vrstvy a začneme odspodu. Důraz v této kapitole klademe na ovladač zařízení (vrstva 2), ale projdeme i zbylé části V/V software a jejich součinnost.

3.2.2 Ovladač přerušení

Přerušení jsou tou méně oblíbenou součásti života. Měly by být schovány hluboko v jádře operačního systému, tak aby o nich věděla co nejmenší část systému. Nejlepší metodou utajení přerušení je pozastavit každý proces, který spustil blok V/V operace, dokud není V/V operace kompletní a dojde k přerušení. Proces se může sám zablokovat voláním například *DOWN* semaforu, *WAIT* na podmíněnou proměnnou, nebo pomocí *RECEIVE* čekat na zprávu. Když dojde k přerušení, obsluha přerušení provede vše co má a odblokuje proces, který operaci odstartoval. V některých systémech provede *UP* semaforu, v jiných zavolá *SIGNAL* pro podmíněnou proměnnou, nebo pošle zprávu zablokovanému procesu. V každém případě je výsledkem zachyceného přerušení odblokování dříve zablokovaného procesu a ten je opět spustitelný.

3.2.3 Ovladač zařízení

Celá část kódu závislá na zařízení se nazývá **ovladač zařízení**. Každý ovladač zařízení obsluhuje zařízení jednoho typu, nebo podobná zařízení ze stejné skupiny. Například je dobré mít jeden ovladač terminálů, i když systém podporuje celou řadu různých typů, které se mírně liší. Na druhé straně hloupý kopírovací terminál a inteligentní grafický terminál s myší se liší příliš na to, aby měli jeden ovladač.

Už dříve jsme si říkali, co dělá řadič. Říkali jsme si, že řadič má jeden nebo více registrů používaných pro zadávání příkazů. Ovladač zařízení zadává tyto příkazy a kontroluje, zda byly správně provedeny. Ovladač disků je tedy jen část operačního systému, která ví, kolik registrů řadič má a k čemu jsou určeny. Jen on sám zná informace o sektorech, stopách, cylindrech, hlavách, pohybu ramen, prokládacím faktoru, času vystavení hlaviček a všech dalších mechanismech, aby disk pracoval správně.

Obecně lze říct, že úkolem ovladače zařízení je přijmout abstraktní požadavky z vrstvy nezávislé na zařízení a dohlédnout, aby byl požadavek proveden. Typickým požadavkem je přečíst blok n. Pokud je zařízení právě volné a přijde požadavek, začne provádět zadaný příkaz okamžitě. Nicméně, pokud je právě zaměstnáno požadavkem, pak je nový požadavek vložen do fronty nevyřízených požadavků, aby byl vyřízen, jakmile to jen bude možné.

První krok je právě přenést V/V požadavek, což v případě disku znamená, přeložit abstraktní požadavek na konkrétní. Pro diskový ovladač to

znamená určit, kde je právě požadovaný blok, zkontrolovat, zda je spuštěný motor, rozhodnout, zda je rameno nastaveno na správném cylindru atd. Zkrátka se musí rozhodnout, které operace řadiče jsou nutné a v jakém pořadí.

Jakmile je rozhodnuto, které příkazy je třeba poslat řadiči, začnou se mu posílat zápisem do registrů řadiče. Některé řadiče jsou schopny v dané chvíli zpracovávat jen jeden příkaz. Jiné řadiče jsou schopny přijmout seznam příkazů a ty samy provedou bez další invence ze strany operačního systému.

Jakmile jsou příkazy, nebo příkaz, zadán řadiči, nastane jedna ze dvou situací. V mnoha případech musí ovladač zařízení čekat, dokud mu řadič neprovede co potřeboval, tak pozastaví sám sebe, dokud nepřijde přerušení a neodblokuje ho. V ostatních případech však operace skončí bez čekání a ovladač zablokování nepotřebuje. Například rolování obrazovky na terminálu vyžaduje zápis několika bytů do registrů řadiče. Není zapotřebí žádný mechanický pohyb a celá operace může být provedena během několika mikrosekund.

V prvním případě je zablokovaný ovladač probuzen přerušením. Ve druhém případě nikdy neusne. A teď opačná cesta, po ukončení operace se musí zkontrolovat chyby. Pokud je vše v pořádku, ovladač může předat data vrstvě nezávislé na zařízení. Nakonec vrátí stavové informace jako hlášení o chybě zpět volajícímu. Pokud jsou ve frontě další požadavky, jeden z nich se může vybrat a začít provádět. Pokud nejsou další požadavky, ovladač se zablokuje čekáním na další požadavek.

3.2.4 Vrstva V/V software nezávislá na zařízení

Ačkoliv je určitá část V/V software specifická pro každé zařízení, velká část je na zařízení nezávislá. Přesná hranice mezi ovladačem a nezávislou vrstvou záleží na systému, protože řadu funkcí, které provádí nezávislá vrstva, lze provést v ovladači a nemusí to být jen z důvodu lepší efektivity. Hlavní funkce nezávislé vrstvy jsou následující:

- jednotné rozhraní pro ovladače zařízení,
- pojmenovávání zařízení,
- ochrana zařízení,
- poskytuje velikost bloku nezávisle na zařízení,
- bufferování (vyrovnávací paměť),
- alokace úložného prostoru na blokových zařízeních,
- přidělení a uvolnění vyhrazených zařízení,
- informace o chybách.

Ačkoli se budeme souborovými systémy zabývat v jiné kapitole, podívejme se trochu na vrstvu nezávislou na zařízení, abychom se lépe seznámili s V/V a jak do toho zapadá ovladač.

Základním úkolem nezávislé vrstvy je provádění V/V funkcí, které jsou obecné pro pro všechna zařízení a nabízí jednotné rozhraní pro uživatelské programy.

Hlavním rysem operačního systému je pojmenovávání objektů, jako jsou soubory a V/V zařízení. Nezávislá vrstva zajišťuje mapování symbolických jmen na odpovídající ovladač. V UNIXu je jméno zařízení, jako například **/dev/tty0**, jednoznačně specifikováno i–nodem pro speciální soubory a tento i–node obsahuje **hlavní číslo zařízení** (major device number), které se používá pro identifikaci odpovídajícího ovladače. Obsahuje také **vedlejší číslo zařízení** (minor device number), které se používá jako parametr ovladače pro identifikaci jednotky pro čtení nebo zápis.

Úzce se jmény souvisí i ochrana. Jak systém brání uživatelům v přístupu k zařízením, kam přistupování povoleno není? Na řadě počítačových systémů není žádná ochrana. Jakýkoliv proces může udělat co chce. Na většině velkých systémů je přístup k V/V zařízení uživatelům zcela odepřen. Na UNIXu se používá docela flexibilní schéma. Speciální soubor odpovídající V/V zařízení je chráněn obvyklými bity *rwx*. Administrátor systému pak může nastavit odpovídající práva pro každé zařízení.

Různé disky mohou mít různé velikosti sektorů. A je úkolem nezávislé vrstvy tento fakt eliminovat a nabízet jednotnou velikost bloku vyšším vrstvám, například spojením několika sektorů do jednoho logického bloku. Takto vyšší vrstvy manipulují s abstraktním zařízením, které používá stejnou velikost bloku nezávisle na fyzické velikosti sektoru. Podobně některá znaková zařízení předávají svá data po bytech, zatím co jiná je doručují ve velkých blocích, jako např. síťová rozhraní. Tento rozdíl se musí také eliminovat.

Bufferování je taky problém pro bloková i znaková zařízení. Pro bloková zařízení obvykle hardware klade důraz na čtení a zápis celých bloků v jedné operaci, ale uživatelský proces může číst a psát po jakýchkoliv dílech. Když uživatelský proces zapíše polovinu bloku, musí data operační systém rozšířit na velikost interní jednotky. Pro znaková zařízení, kdy uživatel může data zapisovat rychleji, než odcházejí, je bufferování nezbytné. Vstup z klávesnice potřebuje taky buffer.

Když se vytvoří nový soubor a naplní daty, musí se alokovat nové bloky. Pro provedení operace potřebuje operační systém seznam nebo bitmapu volných bloků na disku, ale algoritmus pro alokaci je nezávislý na zařízení a provede se nad ovladačem.

Některá zařízení, jako zapisovací CD-ROM, mohou být používána v daném čase jen jediným procesem. Je na operačním systému vyhodnotit požadavek na zařízení a přijmout ho, nebo odmítnout, podle aktuální dostupnosti zařízení. Jednoduchou metodou provedení tohoto požadavku je přímé provedení *OPEN* speciálního souboru pro zařízení. Pokud není zařízení dostupné,

požadavek selže. Uzavření takového zařízení ho pak opět uvolní.

Manipulace s chybami je celkem vzato prováděna ovladačem. Většina chyb je velmi závislá na zařízení a jen ovladač ví, co dělat. Typická chyba je způsobena blokem na disku, který byl zničen a nemůže být znovu přečten. Jakmile se ovladač pokusí několikrát přečíst blok znovu, předá informaci nezávislé vrstvě. Jak se tady s chybou naloží, už na zařízení nezávisí. Když nastane chyba při čtení souboru uživatelem, je vhodné předat chybu volajícímu. Nicméně, pokud dojde k chybě při čtení důležité systémové struktury, jako např. bloky obsahující bitmapu volných bloků, má operační systém na výběr: skončit nebo vypsat chybu na terminál.

3.2.5 Uživatelský V/V software

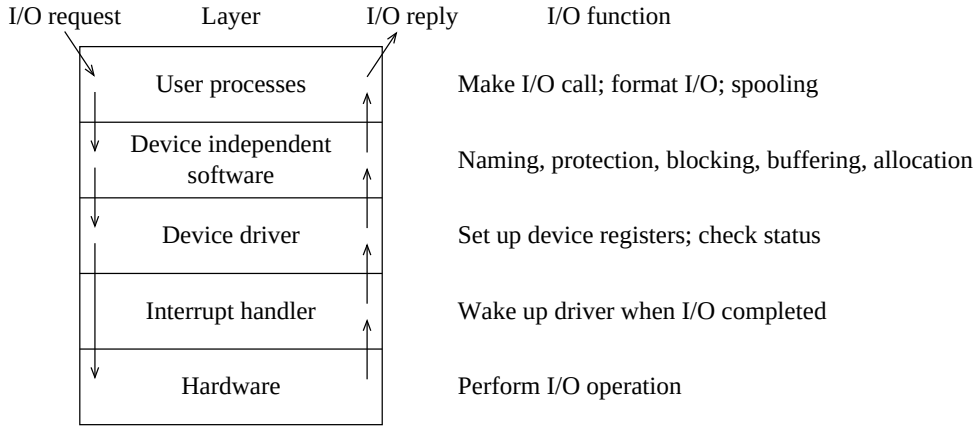
I když je většina V/V software v operačním systému, jeho malá část je v knihovnách připojovaných k uživatelským programům a taky programy běží mimo kernel. Systémová volání, včetně volání V/V operací, se provádí podprogramy z knihoven. Když program v jazyce C volá:

```
count = write( handle, buffer, length );
```

připojí se k programu knihovna s funkcí *write* a v době běhu programu je její binární kód v paměti. Sada těchto podprogramů se jednoduše součástí V/V systému.

I když tyto podprogramy dělají přeci jen něco víc, než jen předávání parametrů do systémových volání, jsou zde V/V podprogramy vykonávající skutečnou práci. Speciální formátování vstupu a výstupu se provádí podprogramy z knihovny. Jako příklad z jazyka C je funkce *printf*, která vezme formátovací řetězec a pravděpodobně i nějaké proměnné jako vstup, vytvoří ASCII řetězec a pak zavolá *write* pro výstup řetězce. Příklad podobné funkce pro vstup je *scanf*, která čte vstup a ukládá ho do proměnných popsaných ve formátovacím řetězci, používajíc stejnou syntaxi jako *printf*. Standardní V/V knihovna obsahuje řadu podprogramů, které vyvolávají V/V operace a běží jako součást uživatelského programu. Ne celá část uživatelského V/V se skládá z knihovních podprogramů. Další důležitou kategorií je spooling (souběžné asynchronní výstupní operace) systém. **Spooling** je způsob manipulace s vyhrazenými zařízeními v multiprogramovém systému. Předpokládejme typické zařízení pro spooling: tiskárnu. Ačkoli je technicky jednoduché povolit uživatelským programům otevřít speciální znakový soubor pro tiskárnu, ale představte si proces, který ho otevřel a pak několik hodin nic nedělá. Žádný další proce nic nevytiskne.

Místo toho vytvoříme speciální proces nazývaný **démon** (daemon) a speciální **adresář pro souběžný tisk** (spooling directory). Abychom vytiskli soubor, musí proces nejprve vytvořit soubor pro tisk a vložit jej do adresáře souběžný tisk. A úkolem démona, jakožto jediného procesu majícího právo přistupovat ke speciálnímu zařízení, je tisknou soubory z adresáře. A



Obrázek 4: Vrstvy V/V software a jejich hlavní funkce.

ochranou speciálního souboru před přímým přístupem uživatelů eliminujeme možnost, že by si někdo mohl soubor otevřít zbytečně dlouho.

Spooling není jen pro tiskárny. Používá se i v jiných situacích. Například pro přenosy souborů po síti se také používají démony. Pro poslání souboru jej uživatel umístí opět do síťového adresáře. Časem si jej démon vyzvedne a pošle. Speciálním případem takového posílání souborů je elektronická pošta. Síť se skládá z miliónů počítačů po celém světě a ty komunikují přes mnoho sítí. Pro odeslání pošty se např. volá program *send*, který vezme dopis k odeslání a uloží jej do adresáře pro odesílání pošty, pro pozdější přeposlání. Celý poštovní systém běží mimo operační systém.

Na obrázku 4 je shrnutí V/V systému, všech jeho vrstev a hlavních funkcí každé z nich. Začíná spodní vrstvou hardware, ovladače přerušení, ovladače zařízení, nezávislé vrstvy a nakonec uživatelský program.

Šipky na obrázku 4 ukazují řízení toku dat. Když se uživatelský program pokouší číst blok ze souboru, je operační systém vyzván voláním. Nezávislá vrstva se podívá do vyrovnávací paměti. Pokud v ní požadovaný blok není, volá ovladač zařízení, aby předal požadavek do hardware. Proces je pak zablokován, dokud není disková operace kompletní.

Když disk skončí, hardware generuje přerušení. Ovladač přerušení zjistí, co se stalo, tedy které zařízení vyžaduje pozornost. Pak převezme status ze zařízení a probudí uspaný proces, aby dokončil V/V požadavek a proces mohl pokračovat.

3.3 Zablokování, uvíznutí (deadlock)

Počítač je plný prostředků, které mohou být v dané chvíli používány jen jedním procesem. Mezi obvyklé případy patří plotr, zapisovačka CD-ROM, pásková mechanika, osvitová jednotka a taky pozice v systémové tabulce

procesů. Mít dva procesy paralelně zapisující na tiskárnu, tak dostaneme blábol. Mít dva procesy současně zapisující do tabulky procesů pravděpodobně povede k havárii. V důsledku čehož má každý systém možnost dočasně zaručit výhradní přístup k určitému prostředku.

V mnoha aplikacích proces vyžaduje výhradní přístup nejen k jednomu zdroji, ale k několika. Představme si marketingovou formu, která se specializuje a výrobu velkých detailních demografických map USA na 1 m širokém plotru. Demografické informace přicházejí z CD-ROMu s výsledky sčítání lidí a dalšími daty. Předpokládejme, že proces *A* požádal o mechaniku CD-ROM a získal ji. O chvíli později proces *B* požádá o plotr a taky jej získá. Nyní proces *A* požádá o plotr a zastaví se čekáním na něj. A nakonec proces *B* požádá o CD-ROM a také se zastaví. Nyní jsou oba procesy blokovány a tak zůstanou navždy. Tato situace se nazývá **zablokování** (deadlock), někdy také říkáme **uvíznutí** (my budeme dále používat první pojem). Zablokování není pro náš systém dobrá věc.

K zablokování může dojít v mnoha případech, kromě požadavků na výhradní přístup. V databázovém systému například může program mít uzamčeny některé používané záznamy, aby zabránil souběhu. Pokud proces *A* uzamkne záznam *R1* a proces *B* záznam *R2* a pak se každý proces pokusí uzamknout právě ten druhý záznam, tak máme opět zablokování. Zablokování může nastat jak v technických, tak i programových zdrojích.

V této kapitole se podíváme na zablokování více zblízka, abychom viděli jak k němu dochází, jak se proti němu bránit a jak se ho vyvarovat. Jako příklady si budeme brát požadavky na fyzická zařízení, jako je páska, CD-ROM a plotr, protože je snadné si je představit, ale principy algoritmů se úplně stejně hodí i pro jiné druhy zablokování.

3.3.1 Prostředky (Resources)

Zablokování může nastat, když je procesu zaručen výhradní přístup k zařízení, souboru, nebo něčemu dalšímu. Abychom se bavili o zablokování co nejobecněji je to možné, budeme nazývat přidělený objekt jako **prostředek**. Prostředkem rozumíme zařízení (páska) nebo i část informace (záznam v souboru). Počítač má obvykle mnoho různých prostředků, o které lze žádat. Některé prostředky mohou být na počítači zastoupeny ve více instancích, jako třeba tři páskové mechaniky. Pokud je k dispozici více identických prostředků, mohou být poskytnuty komukoliv, kdo o ně požádá. Stručně, prostředkem je cokoliv, co může být použito jedním procesem v kterémkoliv okamžiku.

Prostředky jsou dvojího typu: odebíratelné a neodebíratelné. **Odebíratelné prostředky** (preemptable resources) jsou takové, jejichž odebrání procesu, který je vlastní, nebude nijak bolestivé. Příkladem odebíratelného prostředku je paměť. Představme si například systém s 512 KB paměti, tiskárnu a dva procesy velikosti 512 KB, která chtějí tisknout. Proces *A* po-

žádá a dostane tiskárnu, pak začne počítat hodnoty a tisknout. Než ukončí počítání, vyprší jeho časové kvantum a je odložen.

Spustí se proces B a pokusí se neúspěšně získat tiskárnu. Teoreticky jsme v situaci zablokování, protože A má tiskárnu, B má paměť a ani jeden nemůže pokračovat bez prostředku toho druhého. Naštěstí je možné odebrat paměť procesu B odložením na disk a natažením procesu A. Nyní může A pokračovat, tisknout a pak uvolnit tiskárnu. Bez zablokování.

Na druhé straně **neodebíratelné zdroje** (nonpreemptable resources), které nemohou být jeho vlastníkovi odebrány bez toho, že to způsobí chybu. Pokud proces začne tisknout a přenechá tiskárnu dalšímu procesu, dostaneme zkomolený výsledek. Tiskárna je neodebíratelná.

Obecně zablokování způsobují neodebíratelné prostředky. Potencionální zablokování způsobené odebíratelnými prostředky se dá obvykle řešit realokací prostředků z jednoho procesu na jiný. Naše pozornost se tedy zaměří na neodebíratelné prostředky.

Pořadí událostí pro použití prostředku je následující:

1. Vyžádání prostředku.
2. Použití prostředku.
3. Uvolnění prostředku.

Pokud není požadovaný prostředek v okamžiku žádosti dosažitelný, žádající proces je donucen k čekání. V některých operačních systémech je proces automaticky pozastaven, když selže žádost o prostředek a je probuzen, když se prostředek uvolní. V ostatních systémech požadavek selže s chybovým návratovým kódem a je na příslušném procesu, aby chvíli počkal a zkusil to znovu.

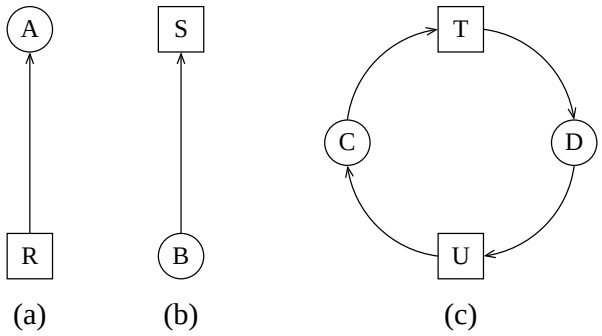
3.3.2 Podstata zablokování

Zablokování můžeme formálně definovat následovně:

Skupina procesů je zablokovaná, pokud každý proces z této skupiny čeká na událost, kterou může vyvolat jen jiný proces z této skupiny

A protože všechny procesy čekají, žádný z nich nikdy nevyvolá událost, která může probudit některý proces ve skupině a všechny procesy budou čekat navěky.

Ve většině případů je událost, na kterou procesy čekají, uvolnění určitého prostředku vlastněného jiným členem ve skupině. Jinými slovy, každý zablokovaný proces ve skupině čeká na prostředky vlastněné jiným zablokovaným procesem. Žádný z procesů nemůže běžet, žádný nemůže uvolnit své prostředky a žádný z nich nemůže být probuzen. Počet procesů a počet a typ prostředků vlastněných a požadovaných, není důležitý.



Obrázek 5: Grafy alokace prostředků. (a) Vlastnit prostředek. (b) Žádat o prostředek. (c) Zablokování.

Podmínky pro zablokování

Aby mohlo dojít k zablokování, musí být splněny 4 podmínky:

1. Podmínka vzájemného vyloučení. Každý prostředek je buď právě vlastněn jedním procesem, nebo je dosažitelný.
2. Podmínka postupného přidělování prostředků. Proces který již vlastní dříve přidělené prostředky, může požádat o další.
3. Podmínka odebrání. Dříve přidělené prostředky nemohou být procesu násilím odebrány. Musí být uvolněny procesem, který je vlastní.
4. Podmínka čekání v kruhu. Musí se vytvořit zřetězený kruh dvou nebo více procesů, kde každý čeká na prostředek vlastněný následujícím procesem v kruhu.

Aby mohlo nastat zablokování, musí existovat všechny čtyři podmínky. Pokud jedna nebo více podmínek chybí, zablokovaní není možné.

Modelování zablokování

Čtyři podmínky se dají modelovat použitím orientovaného grafu. Graf má dva typy uzlů: procesy jako kolečka a prostředky jako čtverečky. Hrana od prostředku k procesu znamená, že daný prostředek byl dříve vyžádán a je procesem vlastněn. Na obrázku 5(a) je prostředek R právě přiřazen procesu A.

Hrana od procesu k prostředku znamená, že proces je právě zablokován čekáním na zdroj. Na obrázku 5(b) čeká proces B na prostředek S. Na obrázku 5(c) vidíme zablokování. proces C čeká na prostředek T, který je právě přiřazen procesu D. Proces D neuvolní prostředek T, protože právě

čeká na prostředek U, přiřazený C. Oba procesy budou čekat navždy. Cyklus v grafu znamená, že je zde zablokování svazující procesy a prostředky do cyklu. Příkladem je cyklus C–T–D–U–C.

Podívejme se teď, jak se dá graf prostředků použít. Představme si tři procesy, A, B a C a tři zdroje R, S a T. Požadavky a uvolňování prostředků těmito třemi procesy jsou na obrázku 6(a)-(c). Operační systém je schopen spustit libovolný odblokovaný proces kdykoliv, může tedy rozhodnout spustit A a nechat jej dokončit práci, pak spustit až do konce B a nakonec C.

Toto pořadí nevede k zablokováním, ale také nemá žádný paralelismus. Kromě požadavků na prostředky, proces počítá a provádí V/V. Pokud se procesy spouští sekvenčně, není zde možnost při čekání jednoho procesu na V/V operace předat CPU jinému procesu. Spouštění procesů čistě sekvenčně tedy není optimální. Na druhé straně, pokud žádný proces neprovádí V/V, kratší dávka jako první je lepší, než round robin. Takže za určitých okolností může být spuštění všech procesů sekvenčně nejlepším řešením.

Předpokládejme ale, že proces nejen počítá, ale provádí i V/V, a round robin je tedy rozumný plánovací algoritmus. Požadavky na prostředky mohou být v pořadí jako na obrázku 6(d). Grafy těchto šesti požadavků v uvedeném pořadí jsou na obrázku 6(e)-(j). Po 4 požadavku se A pozastaví čekáním na S, jak je vidět na obrázku 6(h). V dalších dvou krocích se B i C také pozastaví a dojde k zablokování v cyklu, jako je to na obrázku 6(j).

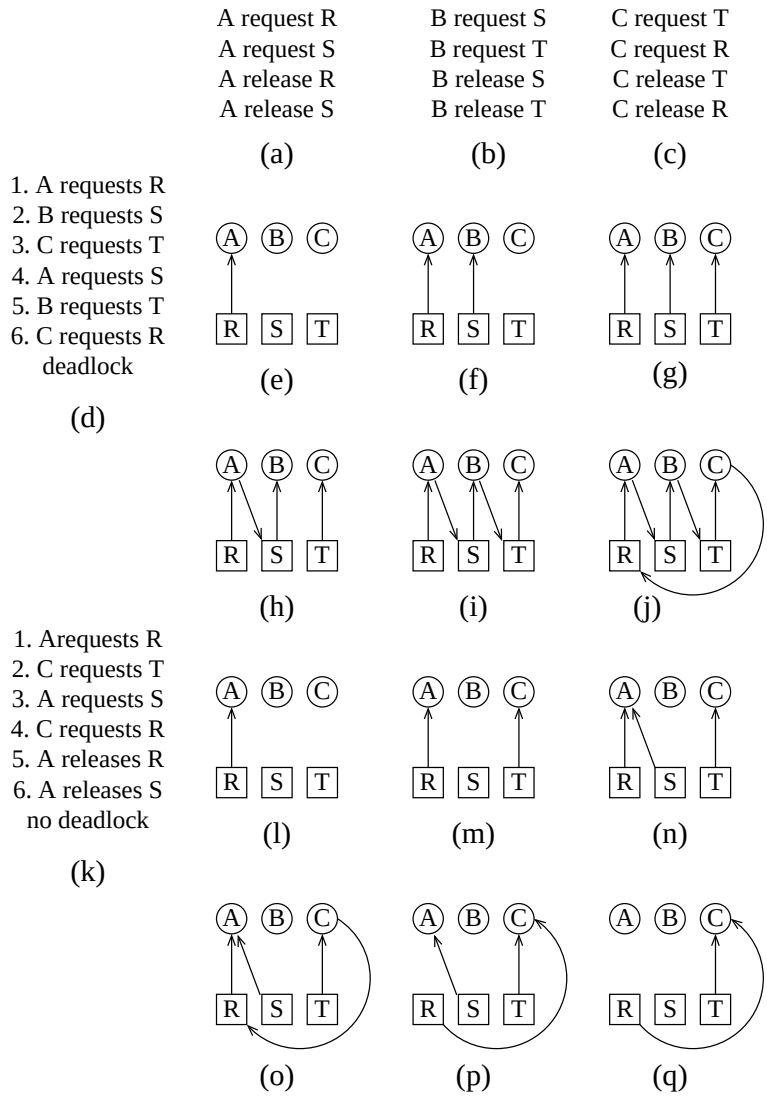
Nicméně jsme již zmínili, že operační systém nemusí vykonávat procesy ve speciálním pořadí. Pokud tedy uspokojování jednotlivých požadavků vede k zablokování, může operační systém jednoduše pozastavit procesy bez uspokojení požadavku a počkat, až budou prostředky k dispozici (bezpečné). Když podle obrázku 6 systém vidí, že nastane zablokování, měl by pozastavit B, místo toho, aby mu přidělil S. Běh samotných procesů A a C seřadí požadavky a uvolňování jako na obrázku 6(k) a ne jako na 6(d). Výsledná sekvence grafů je zachycena na obrázcích 6(l)-(q) a nevede k zablokování.

Po kroku (q) může být procesu B přiděleno S protože A již skončil a C má vše co potřebuje. Jen B se může pozastavit čekáním na T, ale nenastane zablokování. B čeká, dokud neskončí C.

Až později v této kapitole probereme detailně algoritmy pro rozhodování o alokacích, které nevedou k zablokování. To, čemu bychom teď měli rozumět je, že grafy zdrojů a procesů nám dávají možnost znázornění pořadí požadavků a uvolňování, vedoucí k zablokování. Můžeme pěkně krok po kroku zakreslovat požadavky a uvolňování do grafu a po každém kroku můžeme kontrolovat, zda neobsahuje cyklus. Pokud ano, máme zablokování, pokud ne, zablokování není.

K řešení zablokování se dají použít čtyři strategie:

1. Ignorovat problém.
2. Detekce a zotavení.



Obrázek 6: Příklad zablokování a příklad, jak se mu vyhnout.

3. Dynamické zamezení pečlivou alokací prostředků.

4. Prevencí, kdy vhodným návrhem vyloučíme jednu ze čtyř podmínek zablokování.

Všechny metody probereme v následujících sekcích.

3.3.3 Pštrosí algoritmus (Ostrich alg.)

Nejjednodušší přístup je pštrosí algoritmus: strčit hlavu do písku a předstírat, že žádný problém není a nebyl. Různí lidé reagují na tuto strategii odlišně. Matematici shledávají toto řešení jako nepřijatelné a říkají, že zablokování je třeba předcházet za každou cenu. Inženýři se ptají, jak často lze tento problém očekávat, jak často systém zhavaruje z jiných důvodů a jak vážné je zablokování. Pokud k zablokování dojde průměrně jednou za 50 let a systém zhavaruje chybou hardware, chybou kompilace a chybou operačního systému, jednou za měsíc, většina techniků nebude ochotna zaplatit velkou ztrátu výkonnosti nebo pohodlí, aby eliminovali zablokování.

Abychom tento kontrast více upřesnili, UNIX potencionálně trpí zablokováním, které není ani detekováno, natož pak automaticky odbouráno. Celkový počet procesů v systému je určen počtem záznamů v tabulce procesů. A tak je tabulka procesů konečný zdroj. Jestliže selže *FORK*, protože tabulka procesů je plná, rozumným doporučením programu, aby provedl *FORK*, je, počkat náhodnou dobu a zkusit to znovu.

Nyní předpokládejme, že UNIX má 100 položek v tabulce. Je spuštěno deset programů a každý potřebuje vytvořit 12 podprocesů. Jakmile každý proces vytvoří 9 procesů, tak 10 procesů původních a 90 nových zaplní tabulku procesů. Všech 10 procesů teď čeká v nekonečné smyčce a neúspěšně vytváří nové procesy – a máme zablokování. Pravděpodobnost, že se něco takového stane, je naprosto minimální. Měli bychom se vzdát procesů a volání *FORK*, abychom tento problém vyloučili?

Maximální počet otevřených souborů je podobně omezen tabulkou i–nodů a podobný problém může nastat, když se zaplní. Odkládací prostor na disku je také limitovaný prostředek. Prakticky téměř každá tabulka v operačním systému představuje omezený prostředek. Máme je všechny zrušit, protože se může stát, že skupina n procesů může požádat o $1/n$ z každé z nich a pak se pokusí požádat o další?

Přístup UNIXu je ignorovat problém s předpokladem, že většina uživatelů upřednostní příležitostné zablokování před omezením všech uživatelů na jeden proces, jeden otevřený soubor a jeden kus ze všeho dostupného. Kdyby se dalo zablokování vyřešit zadarmo, není o čem mluvit. Ale cena řešení je vysoká, většinou nevyhovujícím omezením procesů, jak brzy uvidíme. Jsme tedy postaveni před nepříjemné rozhodnutí mezi pohodlím a korektností a velkou diskuzi, co je důležitější.

3.3.4 Detekce a zotavení (Detection and recovery)

Druhou strategií je detekce a zotavení. U této techniky nedělá systém nic jiného, než že eviduje požadavky a uvolňování prostředků. Vždy, když je prostředek vyžádán a uvolněn, je aktualizován graf prostředků a je provedena kontrola, jestli se v něm neobjevil cyklus. Pokud cyklus existuje, jeden proces z cyklu je zrušen (killed). Pokud se tím zablokování neodstraní, je zrušen další proces, dokud se smyčka neodstraní.

Poněkud surovější metoda neudržuje graf prostředků, ale periodicky kontroluje, zda nějaký proces není trvale zablokován déle než jednu hodinu. Takový proces je ukončen.

Detekce a zotavení je strategií často používanou na velkých systémech, speciálně dávkových, kde je ukončení procesu a jeho opětovné spuštění přijatelné. Jen je nutno se věnovat pečlivému uvedení modifikovaných souborů do původního stavu, a všemu, kde došlo ke změnám.

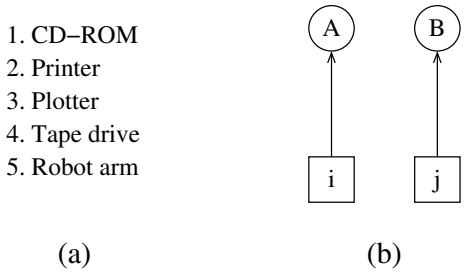
3.3.5 Předcházení zablokování (Deadlock Prevention)

Třetí strategie zablokování je zavedení přijatelných omezení procesům tak, aby bylo zablokování strukturálně (konstrukčně) nemožné. Čtyři základní podmínky zablokování nám dávají vodítko k možnému řešení. Pokud zajistíme, že minimálně jedna z podmínek nebude nikdy splněna, k zablokování nikdy nedojde.

Pokusme se pustit do rozboru podmínky vzájemného vyloučení. Pokud nebude jedinému procesu výhradně přiřazen žádný prostředek, nikdy nedojde k zablokování. Nicméně je dostatečně jasné, že povolit přístup na tiskárnu dvěma procesům současně povede k chaosu. Použitím spoolingu pro tisk může generovat tiskové výstupy více procesům. V tomto modelu je jediným procesem přistupujícím k tiskárně démon. Pokud by démon nikdy nepožadoval další prostředky, máme vyřešeno zablokování pro tiskárnu.

Bohužel, ne všechna zařízení lze obsluhovat spoolingem. Kromě toho, soutěžení o místo na disku pro spooling samotný vede k zablokování. Co se stane, pokud dva procesy zaplní polovinu dosažitelného spooling prostoru a ani jeden z nich ještě není hotov? Jestliže je démon naprogramován tak, že nezačne tisknout, dokud není výstup hotový celý, může i několik hodin čekat, než proces dokončí svůj výstup. Z těchto důvodů se normálně programuje tak, že začne tisknout, až je výstup programu celý. Žádný proces neskončí, pokud je zablokování za disku.

Druhá podmínka zablokování je slibnější. Pokud zabráníme procesu, vlastnícímu nějaké prostředky, čekat na další, můžeme se vyhnout zablokování. Jedna cesta, jak toho dosáhnout je, že si proces vyžádá všechny potřebné prostředky před svým spuštěním. Pokud budou všechny prostředky dostupné, proces by měl mít alokováno vše co potřebuje a může pracovat až do konce. Pokud by jeden nebo více prostředků bylo používáno, nebude nic



Obrázek 7: (a) Očíslované a seřazené prostředky. (b) Graf prostředků.

alokováno a proces zůstane čekat.

S takovým přístupem přijde bezprostředně problém, že mnoho procesů neví, kolik prostředků budou potřebovat, dokud se nespustí a nepoběží. Dalším problémem bude neoptimální využívání prostředků. Například proces si načte data z pásky, bude je několik hodin analyzovat a pak vykreslí výsledek. Pokud musí být všechny zdroje alokovány předem, bude mít proces vyhrazenu pásku i plotr několik hodin

Poněkud odlišnou cestou poručení podmínky postupného přidělování prostředků je vynucení si u každého procesu požadujícího nějaký prostředek, dočasné uvolnění všech prostředků. Jen pokud je požadavek úspěšný, může získat zpět všechny své původní prostředky.

Možnost porušení třetí podmínky odebírání prostředků je ještě méně slibné, než té předchozí. Pokud je tiskárna přiřazena procesu a je uprostřed tisku, tak její násilné odebrání povede ke zmatku.

A zbývá už jen jedna podmínka. Kruhové čekání se dá eliminovat mnoha způsoby. Jednou snadnou možností je pravidlo, že každý proces může mít jen jeden prostředek. Pokud potřebuje další, musí nejdříve uvolnit ten předchozí. Pro procesy, které musí kopírovat velká soubor z pásky na tiskárnu je toto nepřijatelné.

Další možností je globální očíslování všech prostředků, jako na obrázku 7(a). A princip je následující: proces může požadovat libovolný prostředek kdykoliv, ale musí to být v uvedeném pořadí. Proces může požádat o o tiskárnu a pak o páskovou mechaniku, ale nikdy nemůže nejprve požádat plotr a pak tiskárnu.

S tímto pravidlem se graf prostředků nemůže nikdy zacyklit. Podívejme se, proč to platí na příklady dvou procesů z obrázku 7(b). K zablokování může dojít jen pokud A požádá o prostředek j a B požádá o i. Pokud předpokládáme, že i a j jsou rozdílné prostředky, budou mít i rozdílná čísla. Pokud $i > j$, pak nesmí A požadovat j. Pokud $i < j$, pak nesmí B požadovat i. V obou případech nedojde k zablokování.

Pro více procesů to funguje stejně. Ve všech případech je jeden z přiřazených prostředků nejvyšší. Proces vlastnící dané prostředky nebude nikdy

žádat o prostředky přiřazené. Proces buď dokončí svou práci, nebo v horším případě bude žádat o všechny prostředky s vyšším číslem. Pokud skončí, uvolní všechny své prostředky. V této chvíli dostanou další procesy prostředky s nejvyšším číslem a také mohou dokončit práci. Zkrátka, je zde postup, jak se postupně ukončí všechny procesy bez zablokování.

Modifikací toho algoritmu je požadavek, že prostředky mohou být požadovány jen přesně ve vzestupném pořadí a pouze se dohlédne, aby žádný proces nepožadoval prostředek nižší, než právě vlastní. Pokud proces na počátku požaduje prostředek 9 a 10 a pak je oba uvolní, může začít vše znovu a tak není důvod zakázat nyní požadavek na zdroj 1.

Ačkoliv očíslování a seřazení prostředků odstraní problém zablokování, je nemožné najít takové pořadí, které by vyhovovalo všem. Když do prostředků zařadíme tabulku procesů, adresář pro spooling, zamykání databáze a další abstraktní prostředky, počet možných prostředků a rozdílnost používání může být příliš velká na to, aby seřazení mohlo fungovat.

Rozdílné přístupy k prevenci zablokování shrnuje tabulka 2

Tabulka 2: Souhrn přístupů k prevenci zablokování.

Podmínka	**Přístup**
Vzájemné vyloučení	Vše přes spooling
Postupné přidělování prostředků	Vyžádat všechny prostředky předem
Neodebírání	Odebrání prostředků
Čekání v kruhu	Číselné pořadí prostředků

3.3.6 Zamezení zablokování

Na obrázku 6 jsme viděli, jak se vyhnout zablokování, ne impozantními diktátorskými pravidly pro procesy, ale pečlivou analýzou všech požadavků na prostředky, abychom viděli, zda je bezpečné je přidělovat. Otázka je, zda existuje algoritmus, který může vždy předejít zablokování správnou volbou v každém okamžiku? Zodpovědně můžeme odpovědět ano, můžeme předejít zablokování, ale jen pokud budou předem známy určité informace. V této části si probereme možnosti předcházení zablokování pečlivou alokací prostředků.

Bankéřův algoritmus pro jeden prostředek

Plánovací algoritmus, který může zabránit zablokování je známý jako **bankéřův algoritmus** (banker's algorithm). Je modelován jako banka v malém městě se skupinou zákazníků, kterým je zaručen rozsah kreditu. Na obrázku 8(a) máme čtyři uživatele a každému z nich je přidělen určitý počet kreditů. Bankéř ví, že všichni zákazníci současně nebudou potřebovat

Name	Used	Max.
Andy	0	6
Barbara	0	5
Marvin	0	4
Suzanne	0	7

Available: 10

(a)

Name	Used	Max.
Andy	1	6
Barbara	1	5
Marvin	2	4
Suzanne	4	7

Available: 2

(b)

Name	Used	Max.
Andy	1	6
Barbara	2	5
Marvin	2	4
Suzanne	4	7

Available: 1

(a)

Obrázek 8: Tři stavy alokovaných prostředků (a) Bezpečný. (b) Bezpečný. (c) Rizikový.

maximum svých kreditů okamžitě, a tak rezervuje jen 10 jednotek, místo 22. V této analogii je banka operačním systémem, jednotky jsou prostředky počítače a zákazníci jsou procesy.

Zákazníci realizují svůj podnikatelský záměr občasnými půjčkami. V určité chvíli je situace jako na obrázku 8(b). Seznam zákazníků obsahuje již zapůjčené jednotky a maximum dovoleného kreditu. Říkáme tomu stav systému s ohledem na alokaci prostředků.

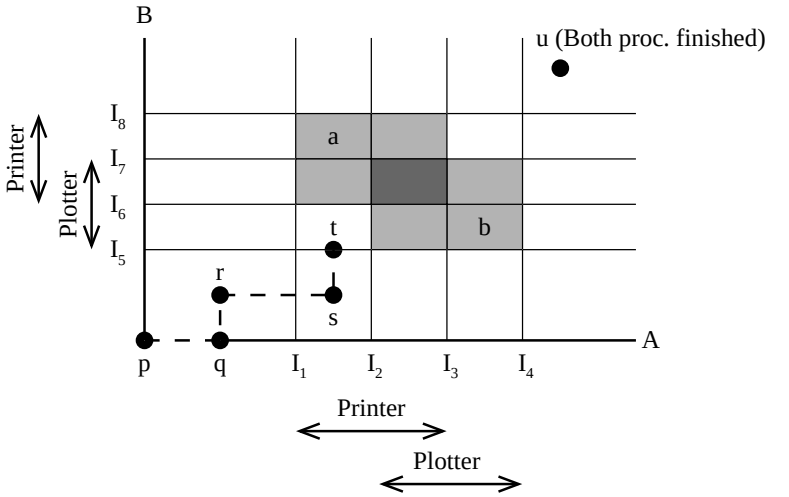
Stav se nazývá bezpečný, jestliže existuje taková posloupnost jiných stavů, která povede k tomu, že každý zákazník si bude moci půjčit až do výše svého limitu. Stav na obrázku 8(b) je bezpečný, protože bankéř může pozdržet všechny požadavky, kromě Marvinova, a tak nechá Marvina dokončit práci a uvolnit všechny 4 své prostředky. Se čtyřmi jednotkami v ruce může bankéř splnit požadavek Barbaře nebo Zuzaně.

Představme si, co by se stalo, kdybychom podle obrázku 8(b) přidělili jeden kredit Barbaře. Budeme mít situaci na obrázku 8(c), která není bezpečná (riziková). Pokud by všichni najednou požádali o svou maximální půjčku, bankéř by nemohl uspokojit požadavky nikomu a měli bychom zablokování. Nebezpečný stav nemusí zákonitě vést k zablokování, protože zákazník nemusí požádat o maximum svého kreditu, ale na to se bankéř nesmí spoléhat.

Je na algoritmu rozhodnout, zda každý požadavek na prostředky, který přijde, povede do bezpečného stavu. Pokud ano, požadavek je uspokojen, jinak je požadavek ponechán na později. Aby bankéř věděl, zda je stav bezpečný, zkontroluje, zda má dostatek prostředků pro uspokojení zákazníků, kteří jsou nejblíže k maximu. Pokud ano, předpokládá se, že půjčka bude splacena a zase se zkontroluje zákazník, který je nejblíže ke svému maximu, atd. Pokud jsou splaceny všechny půjčky, ocitáme se opět v počátečním stavu.

Trajektorie prostředků

Předchozí algoritmus byl navržen s ohledem jen na jeden prostředek. Na



Obrázek 9: Trajektorie dvou procesů.

obrázku 9 vidíme model chování dvou procesů a dvou prostředků, například plotr a tiskárna. Vodorovná osa představuje počet instrukcí vykonaných procesem A a Svislá osa je pro proces B. V čase I_1 požádá A o tiskárnu, v I_2 potřebuje plotr. Tiskárna a plotr budou uvolněny v čase I_3 a I_4. Proces B potřebuje plotr od I_5 do I_7 a tiskárnu od I_6 do I_8.

Všechny body v diagramu reprezentují společné stavy obou procesů. Na počátku je stav p, kdy ani jeden proces nevykonal žádnou instrukci. Pokud se plánovač rozhodne spustit A jako první, dostaneme se do q, kdy A již vykonal určitý počet instrukcí, ale B žádnou. V bodě q se trajektorie otočí do vertikálního směru, což znamená, že byl plánovačem vybrán ke spuštění proces B. V jednoprocesorovém systému musí být všechny cesty vodorovné nebo svislé, nikdy diagonální. Dále je pohyb vždy nahoru nebo doprava, nikdy ne opačně, protože čas nemůže běžet pozpátku.

Když A překročí I_1 na cestě z r do s, kdy je uspokojen požadavek na tiskárnu. Když B překročí bod t, požádá o plotr.

Šedé oblasti jsou mimořádně zajímavé. Oblast a reprezentuje současné používání tiskárny. Tuto oblast je možno zabezpečit podle pravidel pro vzájemné vyloučení. Podobně oblast b představuje společné použití plotru a je také nedostupná.

Pokud někdy vstoupí systém do obdélníka ohraničeného I_1, I_2, I_5 a I_6, dojde nakonec k zablokování při přechodu přes I_2 nebo I_6. V těchto bodech A požaduje plotr a B požaduje tiskárnu, přičemž oba zdroje jsou již přiřazeny. Celý obdélník není bezpečný nesmí se do něj vstoupit. V bodě t je jedinou bezpečnou možností spustit proces A až po I_4. Za touto čarou může vést jakákoliv trajektorie do bodu u.

Obrázek 10: ...

Bankéřův pro více prostředků