Operační systémy 1

Meziprocesová komunikace (IPC)

Petr Krajča

Katedra informatiky
Univerzita Palackého v Olomouci

IPC: Inter-process communication

- procesy oddělené, potřeba kooperace
  - sdílení informací
  - zrychlení výpočtu (rozdělení úlohy na podúlohy)
  - souběžné činnosti
  - modularita
  - oddělení privilegií
- základní kategorie:
  - synchronizace
  - sdílená paměť
  - zasílání zpráv
  - vzdálené volání procedur
- rozlišujeme různé charakteristiky
  - zda komunikují dva příbuzné (mající společného rodiče) nebo zcela cizí procesy
  - zda komunikující proces může jen číst či jen zapisovat data
  - počet procesů zapojených do komunikace
  - zda jsou komunikující procesy synchronizovány, např. čtecí proces čte, až je co číst
  - zda jsou v rámci jednoho systému

Sdílená paměť

- procesy sdílí úsek paměti $\Longrightarrow$ nutná spolupráce se správou paměti
- čtení i zápis, náhodný přístup
- deklarace, že paměť je sdílená + namapování do adresního prostoru
- velikost úseku paměti i adresa zaokrouhleny na násobky stránek paměti (typicky 4 KB)
- paměť může být namapována na různé adresy!!!

Windows

- používá se mechanizmus pro mapovaní souborů do paměti
- `CreateFileMapping`, `MapViewOfFile`
- lze použít i stránkovací soubor

Unix

- `shmget` – vytvoří/najde úsek sdílené paměti s daným klíčem (nastaví oprávnění)
- `shmat` a `shmdt` – namapuje/odmapuje sdílenou paměť z adresního prostoru

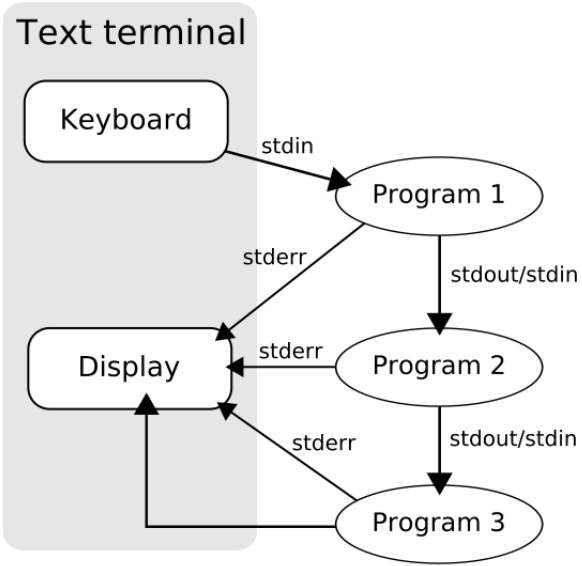
Signály

- mechanizmus podobný přerušení (asynchronní volání)
- základní forma komunikace v unixech
- proces může definovat vlastní handlery těmto signálům
- procesu je možné zaslat jeden z celočíselných signálů
- některé speciální určení (případně nastavené implicitní handlery)
  - SIGINT – ukočení procesu (Ctl+C)
  - SIGQUIT – ukočení procesu (Ctl+/) + Core dump
  - SIGSTOP – pozastavení procesu (Ctl+Z)
  - SIGCONT – pokračování pozastaveného procesu
  - SIGCHLD – změna stavu potomka
  - SIGKILL – nezablokavatelný signál ukončující proces
  - SIGFPE, SIGBUS, ... – oznamování systémových chyb
  - SIGUSR1, SIGUSR2 – uživatelské signály
  - SIGALRM – alarm
  - SIGPIPE – přerušená roura
- race-conditions
- nelze zasílat složitější zprávy

Roury (1/4)

- typická vlastnost unixových OS (ale podpora i ve Windows)
- mechanizmus umožnující jednosměrnou komunikaci mezi procesy
- komunikace dvou procesů (jeden zapisující konec, druhý čtecí konec)
- First-In-First-Out
- umožňuje propojit vstupy výstupy procesů $\Longrightarrow$ kompozice do větších celků
- v shellu: `cat foo.log | grep "11/11/2011" | wc -l`
- využití společně se standardním vstupem (stdin) a výstupem (stdout)
- typické použití:
  - rodičovský proces vytvoří rouru voláním `pipe` (dva popisovače souborů – zápis a čtení)
  - po zavolání `fork()` potomek dědí oba tyto popisovače
  - rodič i potomek zavírají nepotřebné popisovače
  - je možné zapisovat/číst z/do jednotlivý popisovačů souborů
- u procesu lze přenastavit popisovače pro stdin a stdout, aby ukazovaly na konec roury
- rodič může propojit dva potomky (oba dědí popisovače)

Roury (2/4)


Text terminal
Keyboard
stdin
Program 1
stderr
stdout/stdin
Display
stderr
Program 2
stderr
stdout/stdin
Program 3

```c
int fds[2];       // deskriptory souboru pro oba konce roury
pipe(fds);        // vytvori rouru
pid = fork();     // vytvori potomka
if (pid == 0) {
  close(fds[1]); // zavre nepotrebny zapisovaci konec

  // presmerujeme std. vstup na cteci konec roury
  dup2(fds[0], STDIN_FILENO);
  execlp("sort", "sort", NULL); // spusti sort
} else {
  close(fds[0]); // zavre nepotrebny cteci konec

  // otevre zapisovaci konec roury pro zapis
  FILE *stream = fdopen(fds[1], "w");
  fprintf(stream, "foo\nbar\nbaz\n");
  fflush(stream);

  close(fds[1]); // zavre i zapisovaci konec roury
  waitpid(pid, NULL, 0);
}
```

Roury (4/4)

- v Linuxu velikost bufferu 64 KB
- pokud je plný, zapisující proces je pozastaven; pokud je prázdný, čtecí proces je pozastaven
- více čtenářů/písařů $\Longrightarrow$ race-condition (operace nemusí být atomické)

Pojmenované roury (FIFO)

- soubor, který se chová jako roura
- volání a program `mkfifo`
- umožňuje komunikaci nepříbuzných procesů

Pseudo-roury

- systém emulující roury (MS-DOS), ale vytváří mezi-soubory
- `dir | sort | more`
- $\Longrightarrow$ `dir > 1.tmp && sort <1.tmp >2.tmp && more <2.tmp`

Zasílání zpráv (1/3)

- message passing
- obecný mechanizmus komunikace mezi procesy ($\Longrightarrow$ různé varianty)
- vhodný pro počítače se sdílenou pamětí i pro distribuované systémy
- základní operace:
  - send(dest, message)
  - receive(src, message)
- send i receive $\Longrightarrow$ jako blokující/neblokující operace
  - send i receive blokující – synchronizace
  - send neblokující, receive blokující – příjemce čeká na zprávu
  - send i receive neblokující

Adresace

- přímá – vhodné pro kooperující procesy
- nepřímá
  - zprávy jsou zasílány do fronty (mailbox), odkud jsou vyzvedány příjemcem
  - různé varianty 1:1, 1:N, N:1, M:N

Zasílání zpráv (2/3)

- zprávy – hlavička + tělo zprávy $\Longrightarrow$ (odlišnost od volání)
- tělo zprávy: pevná vs. proměnlivá velikost
- hlavička: typ zprávy, zdroj, cíl, délka zprávy, (kontrolní informace, priorita)

Vzájemné vyloučení

- zasíláním zpráv lze implementovat vzájemné vyloučení
- využívá se blokujícího receive
- společná schránka obsahuje žádnou nebo jednu zprávu $\Longrightarrow$ token udávající, že lze vstoupit do kritické sekce
  - pokud je ve schránce jedna zpráva $\Longrightarrow$ doručena právě jednomu procesu, ostatní jsou blokovány
  - pokud je fronta prázdná, všechny (ostatní) procesy jsou blokovány

Zasílání zpráv (3/3)

```
void p() {
  message msg;
  while (1) {
    receive(box, msg);
    // kriticka sekce
    send(box, msg);
  }
}

void main() {
  create_mailbox(box);
  send(box, null);
  par_begin(p(), p(), p());
}
```

Zasílání zpráv v OS (1/2)

POSIX Message Queue

- nepoužívá se často, není součástí std. knihovny (`librt`)
- velikost fronty a zpráv pevná (definovaná při otevření)
- `mq_open`, `mq_send`, `mq_receive`
- v současnosti se používá spíš D-BUS, ZeroMQ

Windows

- událostmi řízený systém
- zprávy zasílané jednotlivým oknům (všechno je okno)
- smyčka událostí součástí funkce `WinMain`

Zasílání zpráv v OS (2/2)

```
int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,
    LPSTR lpCmdLine, int nCmdShow)
{
  MSG msg;
  while(GetMessage(&msg, NULL, 0, 0) > 0)
  {
    TranslateMessage(&msg);
    DispatchMessage(&msg);
  }
  return msg.wParam;
}
```

Další mechanizmy

- Remote Procedure Calls
  - klient volá zástupnou proceduru (stub)
  - zástupná funkce provede převod parametrů a odešle zprávu
  - systém předá zprávu cílovému počítači
  - server zpracuje příchozí zprávu (provede převod parametrů)
  - provede se procedura
  - odpověď je vrácena opačným způsobem
- různé implementace – CORBA, .NET Remoting, Java Remote Method Invocation, XML-RPC, SOAP
- Windows: DDE, COM, clipboard, Mailslots, pojmenované roury (i přes síť)
- (unixové) sockety – jako síťové rozhraní, ale lokální ($\Longrightarrow$ rychlejší)

Remote Procedure Call