Systèmes d'exploitation

IPC

Bibliographie pour aujourd'hui

- Operating Systems Concepts
  - Chapitre 6

Margaret Hamilton

- Américaine
- *MIT Instrumentation Lab*
- Apollo Guidance Computer

Contenu

- Planification
  - Non-préemptif
  - Préemptif
- IPC
  - signal
  - pipe
  - socket

IPC - COMMUNICATION ENTRE PROCESSUS

Processus

- **processus** - une unité d'exécution
- Chaque processus a:
  - Propre espace mémoire virtuel
  - Tableau de descripteurs
- processus de séjour:
  - READY
  - RUNNING
  - WAITING
- Multitasking

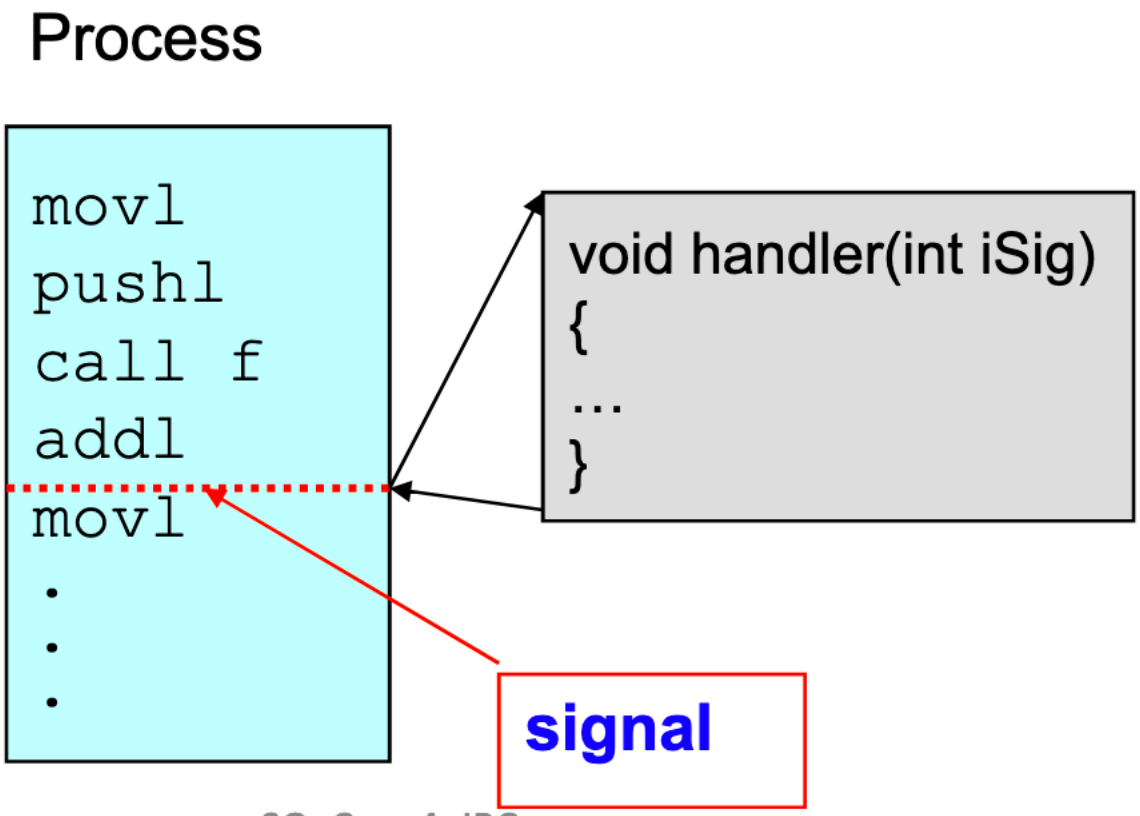
IPC

- signaux (Linux)
- pipe
- mémoire partagé
- socket

Signaux

- Une notification à un processus
- SO peut transmettre des évènements a un processus
- SO découvre un événement (par exemple une interruption)
  - SO arrête le processus (où il est frappé)
  - Le **signal handler** fonctionne
  - Le processus continue là où il en reste

Signaux


Process
movl
pushl
call f
addl
movl
.
.
.
void handler(int iSig)
{
...
}
signal

Liste des signaux - Linux

```
 1) SIGHUP       2) SIGINT       3) SIGQUIT      4) SIGILL
 5) SIGTRAP      6) SIGABRT      7) SIGBUS       8) SIGFPE
 9) SIGKILL     10) SIGUSR1     11) SIGSEGV     12) SIGUSR2
13) SIGPIPE     14) SIGALRM     15) SIGTERM     17) SIGCHLD
18) SIGCONT     19) SIGSTOP     20) SIGTSTP     21) SIGTTIN
22) SIGTTOU     23) SIGURG      24) SIGXCPU     25) SIGXFSZ
26) SIGVTALRM   27) SIGPROF     28) SIGWINCH    29) SIGIO
30) SIGPWR      31) SIGSYS      33) SIGRTMIN    34) SIGRTMIN+1
35) SIGRTMIN+2  36) SIGRTMIN+3  37) SIGRTMIN+4  38) SIGRTMIN+5
39) SIGRTMIN+6  40) SIGRTMIN+7  41) SIGRTMIN+8  42) SIGRTMIN+9
43) SIGRTMIN+10 44) SIGRTMIN+11 45) SIGRTMIN+12 46) SIGRTMIN+13
47) SIGRTMIN+14 48) SIGRTMIN+15 49) SIGRTMAX-15 50) SIGRTMAX-14
51) SIGRTMAX-13 52) SIGRTMAX-12 53) SIGRTMAX-11 54) SIGRTMAX-10
55) SIGRTMAX-9  56) SIGRTMAX-8  57) SIGRTMAX-7  58) SIGRTMAX-6
59) SIGRTMAX-5  60) SIGRTMAX-4  61) SIGRTMAX-3  62) SIGRTMAX-2
63) SIGRTMAX-1  64) SIGRTMAX
```

Envoyer un signal

- Shell
  - kill -numero_signal pid
- POSIX
  - `int kill(pid_t pid, int sig);`
  - `int sigqueue(pid_t pid, int sig, const union sigval value);`

Action sur réception d'un signal

- Action implicite
  - fermer processus (généralement)
- Action utilisateur
  - signal handler
- Ignorance
  - sauf SIGKILL et SIGSTOP

Action Utilisateur

```
int sigaction(int signum,
              const struct sigaction *act, struct
              sigaction *oldact);

struct sigaction {
    void (*sa_handler)(int);
    void (*sa_sigaction)(int, siginfo_t *, void *);
    sigset_t sa_mask;
    int sa_flags;
};
```

Exemple

```c
static void usr2_handler(int signum) {
    /* actions that should be taken when the signal signum
is received */
    ...
}
int main(void) {
    struct sigaction sa;

    memset(&sa, 0, sizeof(sa));

    sa.sa_flags = SA_RESETHAND; /* restore handler to
previous state */
    sa.sa_handler = usr2_handler;
    sigaction(SIGUSR2, &sa, NULL);

    return 0;
}
```

Mask des signaux

```c
int sigemptyset(sigset_t *set);

int sigfillset(sigset_t *set);

int sigaddset(sigset_t *set, int signo);

int sigdelset(sigset_t *set, int signo);

int sigismember(sigset_t *set, int signo);
```

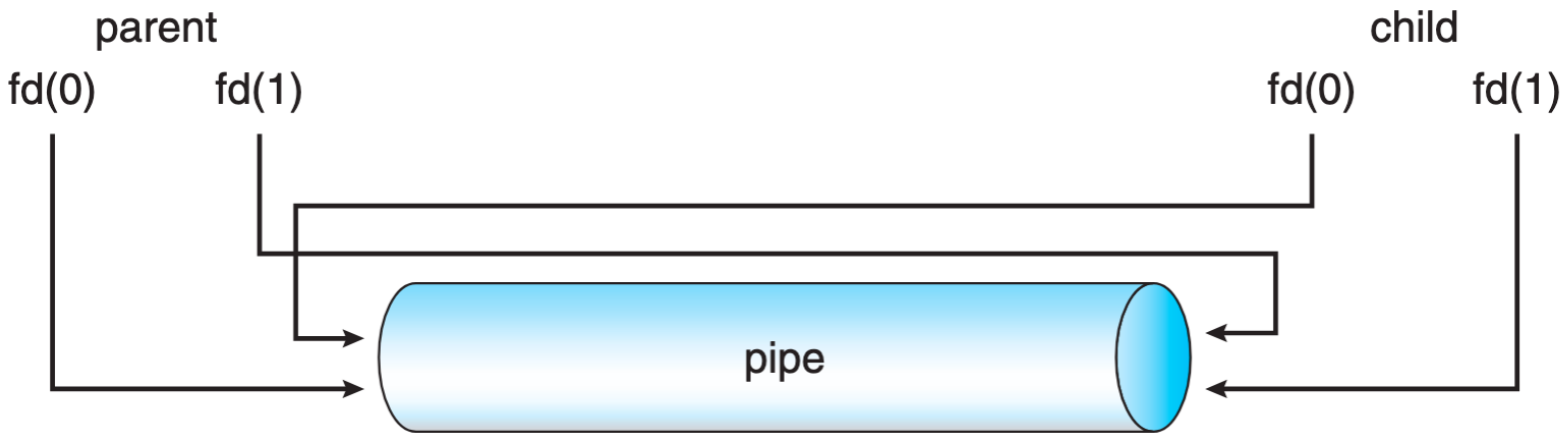
Sd'E
system crunch

PIPE

Anonymes

- Communication entre processus liés (parent / enfant)
- Fonction pipe
  - Retourne deux descripteurs:
    - l'un utilisé pour écrire
    - un autre pour lire

Anonymes



Pipe Exemple Linux

```
int p[2];
int rc;
rc = pipe (p);
if (rc < 0)
{
    perror ("pipe");
}
else
{
    // p[0] <- read from here
    // p[1] -> write here
}
```

Pipe Exemple Linux

```c
int p[2];
int rc, pid;
char buffer[BUFFER_SIZE];
rc = pipe (p);
if (rc < 0) {
    perror ("pipe");
}
else
{
    pid = fork ();
    if (pid == 0) {
        close (p[1]);
        read (p[0], buffer, BUFFER_SIZE);
        // ...
    }
    else
    if (pid > 0) {
        close (p[0]);
        memcpy (buffer, ..., BUFFER_SIZE);
        write (p[1], buffer, BUFFER_SIZE);
        // ...
    }
}
```

Nommés

- mkfifo / mknod
- Utilisé comme un fichier normal
  - open
  - read, write
- Half-duplex (Linux)

Exemple Linux

```
int mkfifo(const char *pathname, mode_t mode);
```

Sd'E
system crunch

AUTRES SYSTEMES IPC

Mémoire partagé

- Chaque processus a sa propre mémoire
  - Une zone de mémoire peut être partagée entre processus
    - shmget / shmat
    - mmap
- Communication sans overhead
  - Pas de traversée dans le noyau
  - Accès à la mémoire
  - Nécessite une synchronisation

socket

- La forme de IPC la plus répandue
  - sockets Unix: similaire aux pipes nommées
- BSD sockets
  - TCP / UDP
  - Cross-machine
  - Synchronisation explicite
- API: flux d'octets fiable
  - socket / connect / accepter / envoyer / recv

Mot clés

- Planification
- Procession en background
- Job
- Systèmes interactifs
- Process
- Temps réel
- Shortest Job First
- Shortest Remaining Job
- First
- Round Robin
- Shortest Process Next
- Priorité
- Priority Inversion
- Priority Boosting
- Starvation
- Completly Fair Scheduler

Questions

Sd'E
system crunch