Lập trình IPC

Bộ môn Hệ thống và Mạng máy tính
Khoa Khoa học và kỹ thuật máy tính

Lập trình trên Linux

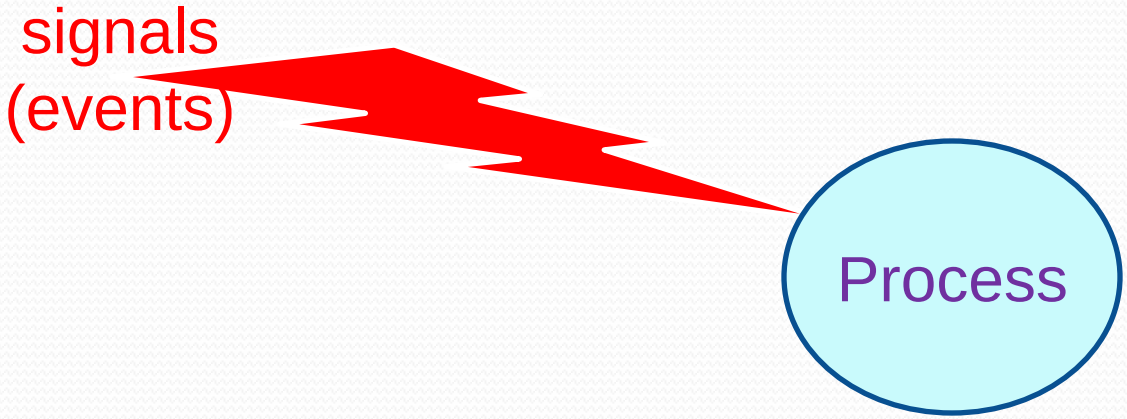
- Lập trình IPC
  - Dùng signal
  - Dùng shared memories

Lập trình trên Linux

- Lập trình IPC
  - Dùng signal
  - Dùng shared memories

Signals

- Dựa vào các sự kiện bất đồng bộ.
- Kernel có nhiệm vụ gửi (deliver) sự kiện đến process
- Các process có thể tự thiết lập các hành vi ứng xử tương ứng với sự kiện nhận được.



Một số signals thường gặp

- SIGKILL
- SIGSTOP
- SIGPIPE
- SIGINT
- SIGQUIT
- …

Tham khảo thêm dùng các lệnh sau

```
$ man 7 signal hoặc $ info signal
$ kill -l
$ more /usr/include/bits/signum.h
```

Các nguồn tạo signal

- Từ kernel
  - Khi xảy ra một số điều kiện về phần cứng (SIGSEGV, SIGFPE)
  - Khi xảy ra điều kiện phần mềm (SIGIO)
- Từ user
  - Tổ hợp phím: Ctrl+C, Ctrl+Z, Ctrl+\
  - Khi user dùng lệnh kill
- Từ một process thực hiện system call kill()

```
#include <sys/types.h>
#include <signal.h>
int kill(pid_t pid, int sig);
```

- Từ lời gọi system call alarm() → tạo ra SIGALRM

Lập trình với signal

```
#include <signal.h>
typedef void (*sighandler_t)(int);
sighandler_t signal(int signum, sighandler_t
  handler);

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

int sighold(int sig);
int sigrelse(int sig);
int sigignore(int sig);
int sigpause(int sig);
```

Lập trình với signal (2)

```
sighandler_t signal(int signum, sighandler_t
  handler);
```

- Thay đổi hành vi của process đối với signal
- Tham số của hàm signal()
  - signum: là số hiệu signal mà bạn muốn thay đổi hành vi (trừ SIGKILL hay SIGSTOP) - dạng số hay symbolic
  - handler: hành vi mới đối với signal, các giá trị có thể là:
    - SIG_DFL: thiết lập lại hành vi về mặc định (default)
    - SIG_IGN: lờ đi (ignore) signal tương ứng
    - Tham chiếu đến hàm xử lý sự kiện (signal-handler) mới do người dùng tự định nghĩa

Lờ đi signal

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
int main() {
  if (signal(SIGINT, SIG_IGN)==SIG_ERR)
  {
    perror("SIGINT\n");
    exit(3);
  }
  while (1);
  return 0;
}
```

Dịch và thực thi

```
$gcc sigign.c -o
sigign
$./sigign
^C
^C
^C
```

Định nghĩa hành vi mới

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
#include <stdlib.h>
void newhandler(int sig) {
  printf("\nI received signal %d",sig);
}
int main() {
   int i=0;
   if (signal(SIGINT,newhandler)==SIG_ERR
     perror("\nSIGINT");
     exit(3);
   }
   while (1);
   return 0;
}
```

Dịch vàthực thi

$gcc sig2.c -o sig2

$./sig2

^C

I received signal 2

^C

I received signal 2

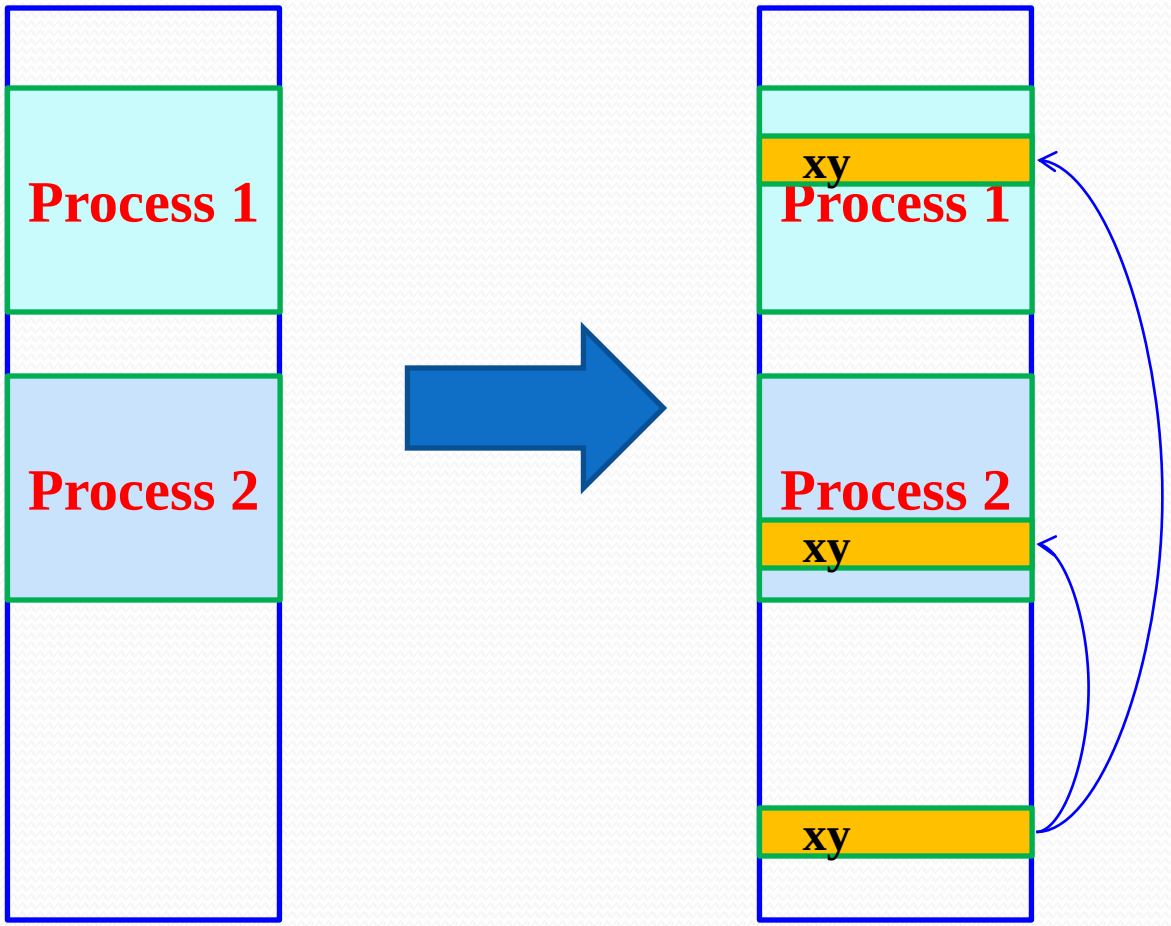
^C

I received signal 2

Lập trình trên Linux

- Lập trình IPC
  - Dùng signal
  - Dùng shared memories

Shared memory



Shared memory

- Có thể theo dõi trạng thái bằng lệnh `ipcs, ipcs -a, ipcs -m`
- Loại bỏ một shared memory bằng lệnh `ipcrm shm shm_id, ipcrm -m shm_id`

```
$ipcs
------Shared Memory Segments --------
key        shmid   owner   perms    bytes    nattch   status
0x00000000 65536   root     644    110592      11       dest

------Semaphore Arrays --------
key         semid   owner   perms    nsems    status

------Message Queues --------
key         msqid   owner   perms    used-bytes     messages
```

Shared memory

- Cho phép nhiều process dùng chung một vùng bộnhớ
  - Kích thước tối thiểu/tối đa của vùng là 1byte/4MB
  - Số vùng nhớ chia sẻ tối đa trong toàn hệ thống: 4096
- Cách sử dụng
  - Vùng nhớ chia sẻ phải được tạo ra trước
  - Process phải gắn vùng nhớ chia sẻ vào không gian địa chỉ của mình trước khi sử dụng.
  - Sau khi dùng xong có thể gỡ vùng nhớ chia sẻ ra khỏi không gian địa chỉ của process

Thao tác với shared memory

- Tạo shared memory

  `shmget()`

- Lấy hoặc thay đổi thuộc tính của shared memory

  `shmctl()`

- Gắn shared memory vào address space của process

  `shmat()`

- Gỡ shared memory khỏi không gian địa chỉ của process

  `shmdt()`

Tạo shared memory segment

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmget(key_t key,int size,int shmflg);
```

- `key`: key tương ứng với shared memory
- `size`: kích thước (tính theo đơn vị byte)
- `shmflg`: tương tự như semflg của semget(), nhưng không có IPC_EXCL

- Ví dụ

```
shm_id = shmget(123, 4096, IPC_CREAT | 0660)
```

Gắn shared memory

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
void *shmat(int shmid,void *shmaddr,int shmflg);
```

- `shmid`: shared memory ID trả về từ hàm shmget()
- `shmaddr`: địa chỉ nơi gắn vùng nhớ chia sẻ
- `shmflg`: SHM_RDONLY (read-only) hoặc 0

Gỡ shared memory

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmdt(void *shmaddr);
```

- `shmaddr`: địa chỉ nơi gắn vùng nhớ chia sẻ (chính là kết quả trả về từ hàm *shmat()*)

Lấy thông tin và thay đổi thuộc tính

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
int shmctl(int shmid,int cmd,struct shmid_ds *buf);
```

- `shmid`: shared memory ID trả về từ hàm shmget().
- `cmd`: IPC_STAT, IPC_SET and IPC_RMID

Ví dụ

- Tạo shared memory 128 bytes
- Hai process dùng chung shared memory
- Process thứ nhất ghi 2 integer vào shared memory
- Process thứ hai đọc từ shared meomory và ghi tổng hai số vào shared memory
- Process thứ nhất đọc tổng và hiển thị ra

Ví dụ

```
int main() {
   int *shm, shmid, k;
   shmid = shmget(IPC_PRIVATE,128,IPC_CREAT|0666);
   shm = (int*) shmat(shmid,0,0);
   if(fork()==0) {      /*child*/
     shm[0]=111;
     shm[1]=999;
     sleep(3);
     pintf("Process %d reads: Sum = %d",
                              getpid(),shm[2]);
     shmdt((void *)shm);
     shmctl(shmid, IPC_RMID, (struct shmid_ds *)0);
```

Ví dụ (cont)

```
    }
    else {          /*parent*/
      sleep(1);
      printf("Process %d writes to shared memory
        ...\n", getpid());
      shm[2]=shm[0]+shm[1];
      shmdt((void *)shm);
    }
    return(0);
}
```

Questions???