Lập trình Hệ thống

Unix Programming

Part 2: C Programming

Nguyễn Quốc Tuấn

Network and Communication System Department
Faculty of Electronics and Communications
UNIVERSITY OF ENGINEERING AND TECHNOLOGY

❑ TIẾN TRÌNH

- Có hai loại tiến trình UNIX *i)* tiến trình liên kết phiên đăng nhập người dùng và *ii)* tiến trình daemon (ở chế độ nền và người dùng không thể điều khiển trực tiếp).

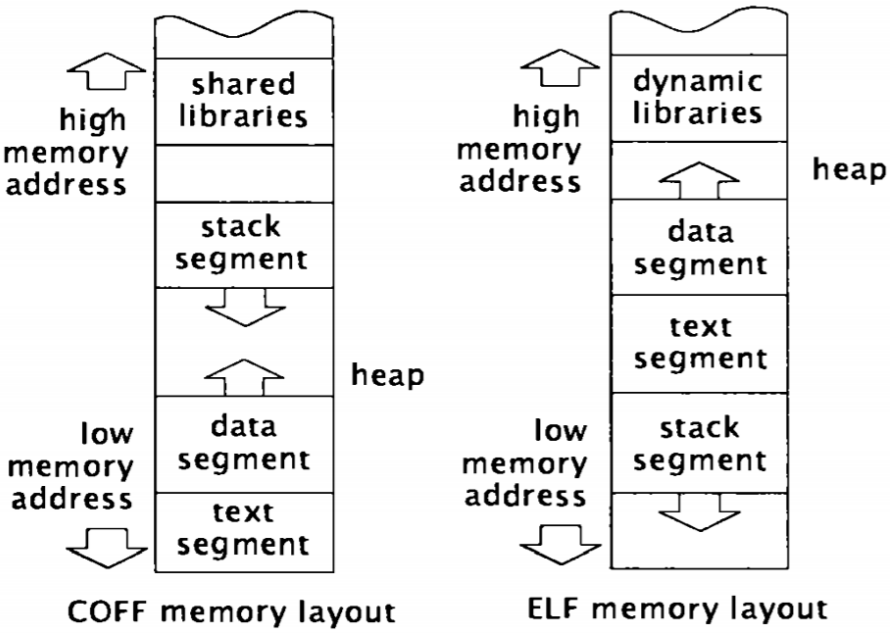
- Mỗi tiến trình có không gian địa chỉ riêng với 3 segment

+ Text chứa mã chương trinh-các lệnh vận hành được

+ Data chứa biến và dữ liệu xử lí khác

+ Stack lưu trữ thông tin như các tham số, địa chỉ trở về,,

- Hai loại tiến trình COFF và ELF sử dụng bộ nhớ ..



- *Dù thứ tự các phân đoạn bộ nhớ khác nhau giữa hai định dạng, nhưng cả hai định dạng đều có phân đoạn text, dữ liệu và ngăn xếp, phân đoạn thư viện chia sẻ tĩnh cho COFF và ELF và động cho ELF. Thuật ngữ text được sử dụng để chỉ lệnh của một chương trình.*
- *Phân đoạn dữ liệu đã khởi tạo được tải từ file chạy trên đĩa và dữ liệu chưa khởi tạo (gọi là BSS) đảm bảo không được lấp đầy trước khi bắt đầu chương trình*

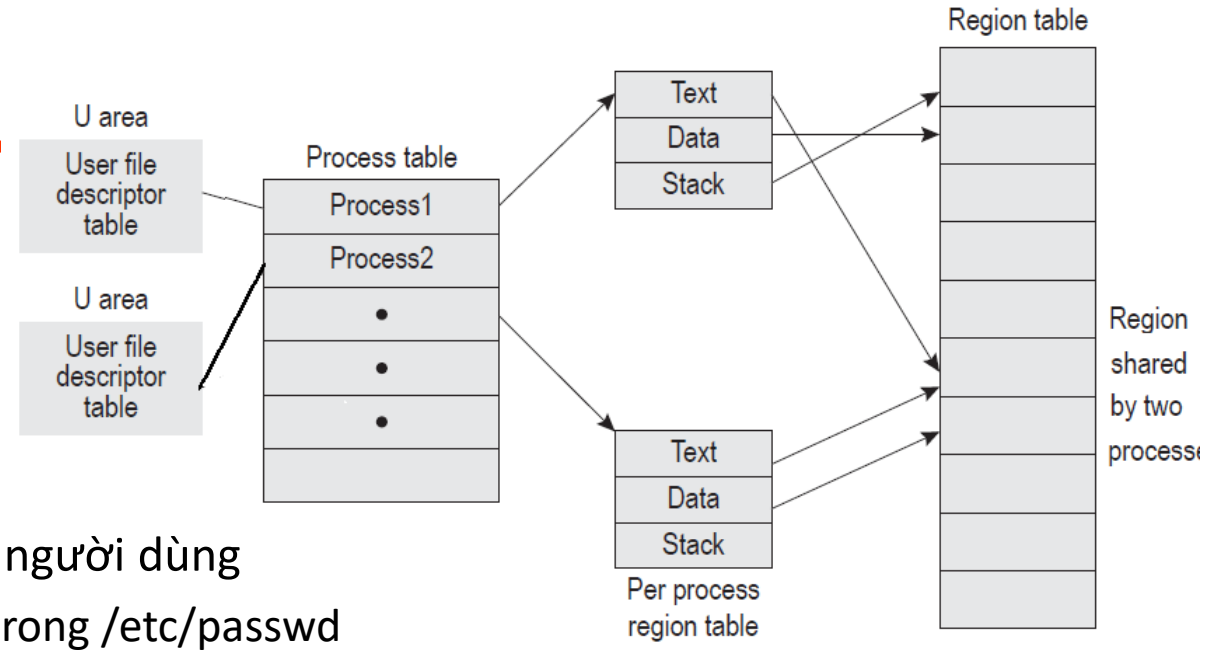
❑ TIẾN TRÌNH

- **Cấu trúc tiến trình** chứa các thông tin cần thiết quản trị tiến trình

- Bảng tiến trình chứa các mục nhập tiến trình - khối điều khiển tiến trình (PCB) với:
  + Trạng thái tiến trình: ready, running, waiting hay mode zombie ..
  + Thông tin nhận dạng tiến trình chứa 3 tham số : PID/UID/PPID
  + Bộ đếm chương trình: Chứa địa chỉ lệnh tiếp theo sẽ được vận hành
  + Các thanh ghi CPU
  + Thông tin lập lich CPU: Chứa giải thuật để lập lịch CPU
  + Thông tin quản trị bộ nhớ: Vùng bộ nhớ liên quan đến tiến trình
  + Thông tin Accounting: như số tiến tính, số jobs, CPU time
  + Thông tin trạng thái vào ra: danh sách thiết bị I/O, trạng thái mở file để xử lí ..
- Vùng người dùng
  + Thông tin UID/PWD/Time/Signal/Data I-O/Err ..
  + File table: vị trí file cho thao tác đọc/viết tiếp, quyền được gán cho
  + Mô tả file giúp theo dõi file được bở bởi tiến trình
- Bảng vùng cho mỗi tiến trình …

C Programming

❑ TIẾN TRÌNH



- **Cấu trúc tiến trình**

Mỗi tiến trình liên quan đến người dùng

- Tên truy cập và mật khẩu trong /etc/passwd

  tuannq: x: 13583: 50: NQ Tuan: /horne/tuannq: /bin/bsh

- Các file chương trình

  -rwxrwxrwx 2 tuannq tuannq 12653 Oct 17 14:12 vdu

- Tiến trình của người dùng tuannq trên máy

  $ps -f -cuser,pid, ppid, pgid, ses, tty, args

  USER PID PPID PGID SESSION TTY COMMANG

  tuannq 1014 505 1014 505 ptx/16 -bsh

  tuannq 1054 1014 1053 505 ptx/16 lpt –deng_lư2

❑ TIẾN TRÌNH

- **Nhận dạng UID/GID**

- Lệnh đọc uid

```
#include <unistd.h>
#include <sys/types.h>
uid_t getuid(void);
uid_t geteuid(void);
```

- Lệnh đọc *gid*

```
#include <unistd.h>
#include <sys/types.h>
gid_t getgid(void);
gid_t getegid(void)
```

- Khi một chương trình được vận hành, nó thêm 2 số nhận dạng effective UID/GID xác định quyền truy cập trong khi vận hành

```
#include <unistd.h>
#include <sys/types.h>
main()
(
    printf("UID = %d, GID = %d\n", getuid(), getgid()) ;
   printf("EUID = %d, EGID = %d\n", geteuid(),
getegid());
   return 0;
}

$ a.out printuid
UID = 500, GID = 50
EUID = 500, EGID = 50
```

❑ TIẾN TRÌNH

- Thông tin mỗi người dùng được xác định qua cấu trúc trường passwd định nghĩa trong thư viện <pwd.h> có dạng

```
struct passwd {
    char  *pw_name        // tên user
    char  *pw_passwd      // mật khẩu
    uid_t  pw_uid         //user ID
    gid_t  pw_gid         // group ID
    char  *pw_comment     // Comment
    char  *pw_dir         // home dir
    char   *pw_shell      // shell cấp
}
```

- Có một số hàm thư viện có sẵn để truy xuất thông tin từ /etc/passwd.

```
#include <pwd.h>
struct passwd *getpwent(void);                  return pointer tới next passwd struct
struct passwd *getpwuid(uid_t uid);             tìm /etc/passwd để match user ID
struct passwd *getpwnam(const char *name);      tìm /etc/passwd để match user name
void  setpwent(void);                           cho phép lặp lại tìm kiếm
void  endpwent(void)l                           đóng passwd file mỗi khi dứt tiến trình
struct passwd *fgetpwent(void);                 return pointer tới next passwd struct
```

❑ TIẾN TRÌNH

- Ví dụ: **$ cat pw.c**

```
#include <pwd.h>
#include <stdio.h>
int main(int argc, char *argv[])
{   struct passwd = *pw;
    if (argc != 2) exit(-1);
    pw = getpwnam(argv[1]);
    if (pw != 0)  {   printf(" Chi tiết cho : %s \n", argv[1]);
          printf(" pw_passwd = %s \n", pw → pw_passwd);
          printf(" pw_uid       = %d \n",pw → pw_uid);
          printf(" pw_gid       = %d \n", pw → pw_gid);
          printf("pw_comment=%s \n", pw → pw_comment);
          printf(" pư_dir         = %s ưn", pw → pw_dir);
          printf(" pw_shell      = %s \n",, pw → pw_shell);
         }
}
```

❑ TIẾN TRÌNH

- **TIẾN TRÌNH**

Compiler

$ cc –o pw pw.c

$ ./pw qtuan

Chi tiet cho qtuan

Pw_passwd	*= x*	*// có mật khẩu*
Pw_uid	*= 13583*	
Pw_gid	*= 50*	
Pw_comment	*= NQ Tuan*	
Pw_dir	*= /home/qtuan*	
Pw_shell	*= /bin/bsh*	

Hàm crypt(S) có thể được sử dụng để mã hóa một chuỗi text được truyền dưới dạng đối số. Khi login(M), nó vô hiệu hóa vọng text đến thiết bị đầu cuối, yêu cầu người dùng nhập mật khẩu và gọi crypt(S) để mã hóa mật khẩu đã nhập. Sau đó so sánh với mật khẩu được mã hóa được lưu trữ trong /etc/shadow để kiểm tra xem mật khẩu có đúng hay không và do đó liệu người dùng có thể đăng nhập hay không.

❑ TIẾN TRÌNH

- Với mỗi groupID có một tên tượng trưng tương ứng được lưu trữ trong file nhóm /etc/group. Cấu trúc của group

```
struct group   {
    char    *gr_name        //tên group
    char    *gr_passwd      //không dùng
    gid_t    gr_gid         // gid
    char    *gr_mem         //list all user
```

Hàm setgrent sẽ mở file nhóm nếu nó chưa mở và rewind. Hàm getgrent đọc mục nhập tiếp theo từ file nhóm (mở file trước nếu nó chưa mở). Hàm endgrent đóng file nhóm.

- Có một số hàm có sẵn trong thư viện <grp.h> cho phép truy cập vào file /etc/group.

```
#include <grp.h>
struct group *getgrgid(gid_t gid);
struct group *getgrnam(const char *name);
```

return: pointer if OK, NULL on error

```
#include <grp.h>
struct group *getgrent(void);
```

Returns: pointer if OK, NULL on error or end of file

```
void setgrent(void);
void endgrent(void);
```

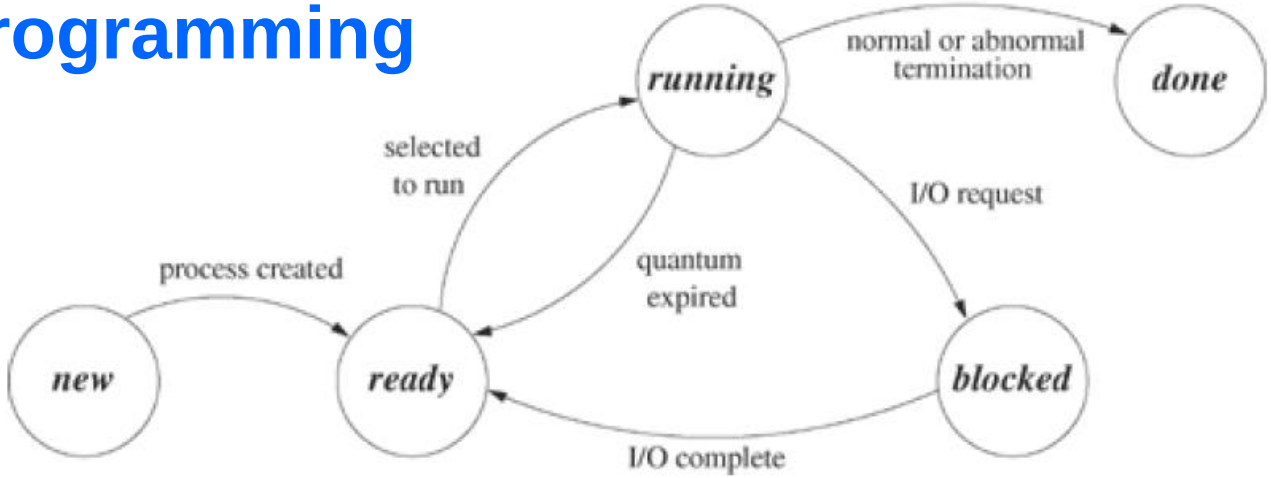
❑ TIẾN TRÌNH

- Chương trình minh họa sử dụng hàm thư viện ***getgrgid ()*** - chuyển ID nhóm số làm đối số, nó truy xuất tên biểu tượng của nhóm.

```
#include <grp.h>
main(int argc, char *argv[])
{
      struct group *grp
      if (argc != 2)
            exit (-1) ;
     grp = getgrgid(atoi(argv[1]) ;
     printf("group name= %s\n", grp->gr_name) ;
}
```

```
$ a.out 101
group name = tuannq
```

❑ TIẾN TRÌNH



- Sơ đồ trạng thái cho hệ điều hành đơn giản:
- Chuyển đổi ngữ cảnh là loại bỏ tiến trình khỏi trạng thái đang chạy và thay thế bằng tiến trình khác
- Ngữ cảnh tiến trình là *thông tin* mà hệ điều hành cần về tiến trình và môi trường của nó để khởi động lại nó sau khi chuyển đổi ngữ cảnh. Rõ ràng, mã vận hành, ngăn xếp, thanh ghi và bộ đếm chương trình là một phần của ngữ cảnh, cũng như bộ nhớ được sử dụng cho các biến tĩnh và động.
- Để có thể khởi động lại tiến trình một cách minh bạch, hệ điều hành theo dõi trạng thái của chương trình I/O, nhận dạng người dùng và quy trình, đặc quyền, thông số lập lịch, thông tin acount và quản lý bộ nhớ và các tài nguyên khác, chẳng hạn như các khóa do tiến trình giữ…

❑ TIẾN TRÌNH

- **Hệ Điều Hành** vận hành NHIỀU tiến trình do vậy các tiến trình mới (con) có thể được tạo ra để chạy. Khi tiến trình mới thoát ra nó quay trở mẹ
- Khuôn dạng: *#include <unistd.h>*

*pid_t **fork**(void);*

Returns: 0 in child, process ID of child in parent, −1 on error

Tiến trình mới được là bản sao của tiến trình cha/mẹ, chỉ có một số khác biệt. Tiến trình con kế thừa nhiều thuộc tính từ cha mẹ bao gồm:

- Biến môi trường.
- Xử lý báo hiệu.
- Các đoạn bộ nhớ dùng chung đính kèm.
- ID nhóm tiến trình.
- PWD với $PATH liên quan.
- Mặt nạ tạo mode file.
- UID/EUID và được lưu.
- GID/EGID và được lưu.
- Điều khiển đầu cuối.

- Tiến trình con có một ID tiến trình duy nhất.
- Tiến trình con có một ID tiến trình mẹ khác.
- Tiến trình con có bản sao riêng của các bộ mô tả tệp của cha mẹ.
- Không có báo hiệu nào đang chờ xử lý ở tiến trình con

❑ TIẾN TRÌNH ME/CON

```
main()
{    int pid ;
     print f ("my pid = %d \n", getpid () );
     pid = fork () ;
     if (pid == -1)
          pexit ("fork")
     else if (pid == 0)
    {    printf("I am the child, my pid = %d\n", getpid()) ;
    }   else
    {   printf("I am the parent, my pid = %d\n", getpid()) ;
    }
}
```

```
$ a.out
my pid = 701
I am the child,  my pid = 702
I am the parent, my pid = 701
```

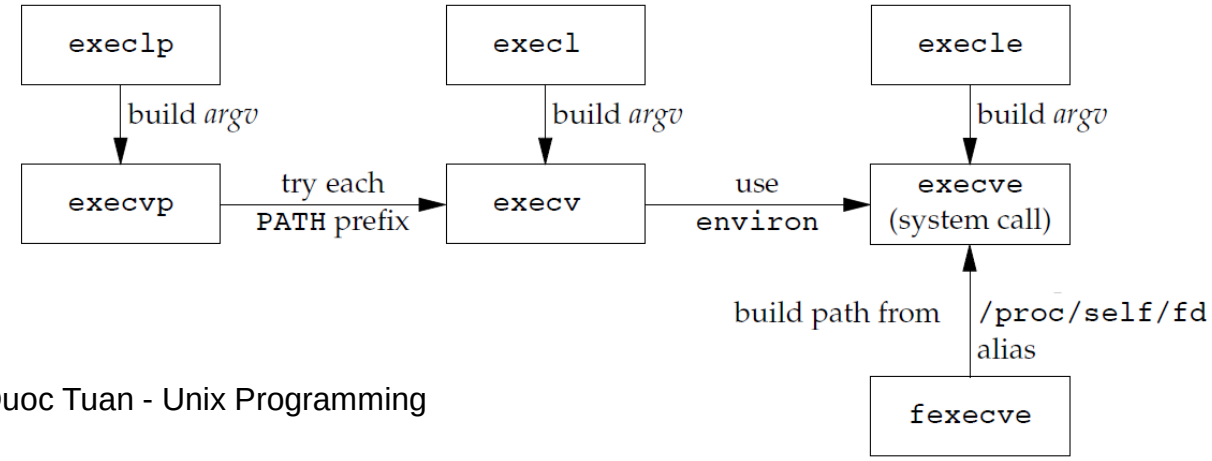
- Mỗi lần fork() được gọi, kernel đảm bảo tiến trình con là một bản sao của tiến trình mẹ, nghĩa là nội dung của code, dữ liệu, ngăn xếp và các đối tượng bộ nhớ khác được sao chép từ tiến trình mẹ sang cho con.

❑ TIẾN TRÌNH ME/CON

- Để vận hành chương trình con cần thay thế code chương trình mẹ, dữ liệu, ngăn xếp và các đối tượng bộ nhớ khác, tiến trình được gọi một trong hàm exec() bên dưới:

```
#include <unistd.h>
int execl(const char *pathname, const char *arg0, ... /* (char *)0 */ );
int execv(const char *pathname, char *const argv[]);
int execle(const char *pathname, const char *arg0, /* (char *)0, char *const envp[] */ );
int execve(const char *pathname, char *const argv[], char *const envp[]);
int execlp(const char *filename, const char *arg0, ... /* (char *)0 */ );
int execvp(const char *filename, char *const argv[]);
int fexecve(int fd, char *const argv[], char *const envp[]);
```

All seven return: −1 on error, no return on success



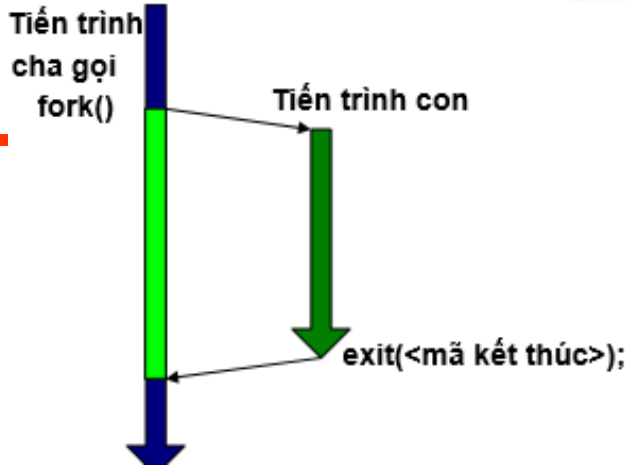
C Programming

❑ TIẾN TRÌNH ME/CON

Chương trình với execle <> execlp
Đầu tiên gọi execle với tên đường dẫn, tên chương trình và môi trường cụ thể. Lệnh gọi tiếp theo execlp sử dụng tên tệp và chuyển môi trường gọi sang chương trình mới.

- Ví dụ:

```
#include "apue.h"
#include <sys/wait.h>
char *env_init[] = { "USER=unknown", "PATH=/tmp", NULL };
Int main(void)
{   pid_t pid;
    if ((pid = fork()) < 0) {
        err_sys("fork error");
    }  else if (pid == 0) {          /* specify pathname, specify environment */
        if (execle("/home/sar/bin/echoall","echoall", "myarg1","MY ARG2", (char *)0, env_init) < 0)
            err_sys("execle error");   }
    if (waitpid(pid, NULL, 0) < 0)
        err_sys("wait error");
     if ((pid = fork()) < 0) {
          err_sys("fork error");
     } else if (pid == 0) {          /* specify filename, inherit environment */
          if (execlp("echoall", "echoall", "only 1 arg", (char *)0) < 0)
               err_sys("execlp error");        }
     exit(0);
}
```



❑ TIẾN TRÌNH ME/CON

- Có hai cơ chế để parent nhận thông báo child tồn tại.

  + Đầu tiên nhờ báo hiệu SIGCHLD báo tiến trình con đã chấm dứt → dùng để gọi waitpid

  + Thứ hai các hàm cho phép tiến trình parent nhận được thông báo về child:

  #include <sys/wait.h>

  *pid_t wait(int *statloc);*

  *pid_t waitpid(pid_t pid, int *statloc, int options);*

  Both return: process ID if OK, 0 (see later), or −1 on error

- Wait() - Cuộc gọi hệ thống sẽ tạm dừng người gọi đến khi một trong child kết thúc hoặc child đang được theo dõi dừng lại do nhận được báo hiệu.

- Hàm ***wait*** có thể chặn người gọi cho đến khi tiến trình con kết thúc, trong khi đó, waitpid có một tùy chọn để tránh khỏi blocking
- Hàm ***waitpid*** không đợi tiến trình con nào kết thúc trước; nó có một số tùy chọn kiểm soát tiến trình mà nó chờ đợi.

C Programming

- Giá trị trả statloc có thể được đánh giá bằng cách sử dụng các hàm macro sau:
  + WIFEXITED(status): *// True nếu status mà child chấm dứt bình thường*
  + WEXITSTATUS(status) *// Lấy 8 bit thấp của đối cho exit(S) hay _exit(S)*
  + WIFSIGNALED(status)* *// True nếu status mà child chấm dứt bất thường (thu số báo hiệu)*
  + WTERMSIG(status) *// Lấy số báo hiệu gây ngắt child*
  + WCOREDUMP(status) *// True nếu core file của tiến trình ngừng phát ra*

❑ **TIẾN TRÌNH ME/CON**

```
#include <sys/types.h>
#include  <sys/wait.h>
main()
{      int pid, statloc ;
       if ((pid =fork()) == -1)  pexit("fork");
       if (pid == 0)                              // child
       {   Sleep(5) ; exit(Ox7f); }
       else {   wait(&statloc) ;                  // parent
            if (WIFEXITED(statloc))
                 printf("Child exited normally with status Ox%x\n", WEXITSTATUS(statloc));
            else
                 printf("Child did not exit normally\n") ;
            }
}
```

C Programming

❑ TIẾN TRÌNH ME/CON

Vận hành tiến trình con

Chương trình

- Đọc command từ đầu vào chuẩn.
- fork() tạo một tiến trình mới và
- Vận hành (process_command)

trong đó chờ child chấm dứt.

```
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <string.h>
#define CMDSIZ 32 ;
extern char **environ
int main(int argc, char *argv[] )
{   int logout=O, cmdsiz;
    char cmdbuf[CMDSIZ] ;
    while(!logout)
    {
        write (1, "myshell> ", 9) ;
        cmdsiz = read(0, cmdbuf, CMDSIZ) ;
        cmdbuf[cmdsiz-1] = '\0' ;
        if (strcmp("logout", cmdbuf) == 0)
            ++logout ;
        else
            process_command(cmdbuf) ;
     }
}
```

❑ TIẾN TRÌNH ME/CON

Vận hành tiến trình con

```
void process_command(char *cmdbuf)
(    - int pid, sloe ;
     pid = fork () ;
     if (pid < 0) write(2, "no more processes\n", 18) ;
     else if (pid == 0)  {          /* child */
          if (execle(cmdbuf, (Char *)0, environ) < 0):
          pexit ("child") ;
     }  else  {                     /* parent */
          wait(&sloc) ;             /* can determine exit status, see wait(S) */
     }
}
```

Trong hàm process_ command (), lệnh gọi fork() tạo tiến trình con
Tên chương trình chạy được nhập từ bàn phím tại chương trình chính đặt vào cmdbuf[ ].
Hàm exec (S) để thực thi tiến trình mới.
Lệnh chờ (wait(S) sẽ block (ngẽn) cho đến khi tiến trình con thoát ra.

❑ TIẾN TRÌNH

▪ Định hướng lại tiến trình con

- POSIX xử lý I/O độc lập với thiết bị thông qua bộ mô tả tệp. Sau khi nhận được một bộ mô tả file mở qua một lệnh gọi như "open" hoặc "pipe", chương trình có thể thực hiện đọc hoặc ghi, sử dụng xử lý được trả về từ cuộc gọi. Chuyển hướng cho phép chương trình gán lại một xử lý đã được mở cho một file chỉ định tới một file khác.
- Ba hàm dưới đây phục vụ định redirect

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int makeargv(const char *s, const char *delimiters, char ***argvp);
int parseandredirectin(char *s);
int parseandredirectout(char *s);
```

- Hàm parseandredirectin tìm biểu tượng "<". Nếu tìm thấy, chương trình sẽ thay thế nó bằng xâu kết thúc (Sử dụng strtok để loại bỏ các khoảng trống và tab ở đầu và cuối chỉ để lại tên file để định hướng lại). Hàm parseandredirectout hoạt động tương tự.
- Hàm *makeargv* trả về số lượng mục trong mảng đối số lệnh. Về mặt kỹ thuật, một lệnh rỗng không phải là một lỗi và một shell sẽ bỏ qua nó mà không in thông báo cảnh báo. Đối với các dòng lệnh phức tạp hơn như chuyển hướng và đường ống, phần lệnh trống được coi là lỗi.

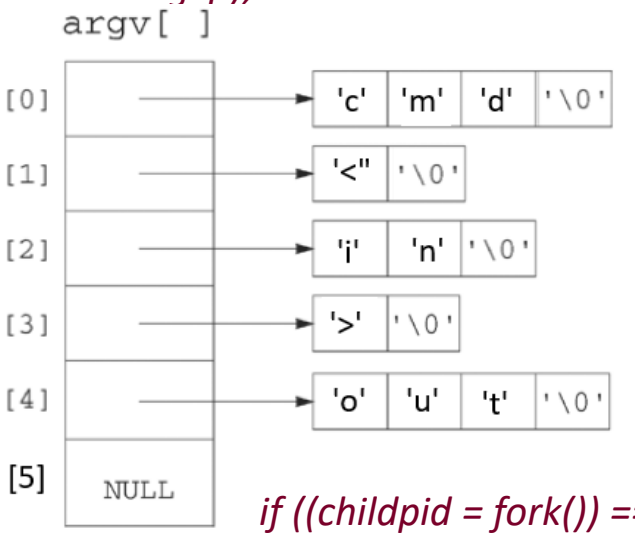
C Programming

❑ TIẾN TRÌNH

- **Định hướng lại tiến trình con**

Ví dụ chương trình vận hành
$ a.out cmd < in > out

```
#include <errno.h>
#include <stdio.h>
#include <unistd.h>
int makeargv(const char *s, const char *delimiters, char ***argvp);
int parseandredirectin(char *s);
int parseandredirectout(char *s);
void executecmd (char *incmd)
Int main(void)
{    pid_t childpid;
     char inbuf[MAX_CANON];
     int len;
     for( ; ; ) {
          if (fputs(PROMPT_STRING, stdout) == EOF)
               continue;
          if (fgets(inbuf, MAX_CANON, stdin) == NULL)
               continue;
          len = strlen(inbuf);
          if (inbuf[len - 1] == '\n')
               inbuf[len - 1] = 0;
          if (strcmp(inbuf, QUIT_STRING) == 0)
               break;
```




```
          if ((childpid = fork()) == -1)
               perror("Failed to fork child");
          else if (childpid == 0) {
               executecmd(inbuf);
               return 1;
          } else
               wait(NULL);
     }
     return 0;
}}
```

❑ TIẾN TRÌNH

- **Định hướng lại tiến trình con**

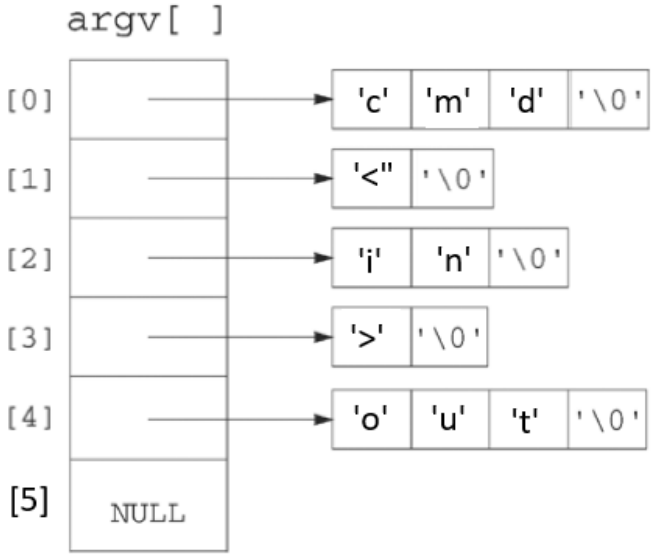
```
void executecmd(char *incmd)
{   char **chargv;
    if (parseandredirectout(incmd) == -1)
          perror("Failed to redirect output");
     else if (parseandredirectin(incmd) == -1)
          perror("Failed to redirect input");
     else if (makeargv(incmd, " \t", &chargv) <= 0)
          fprintf(stderr, "Failed to parse command line\n");
      else {
           execvp(chargv[0], chargv);
           perror("Failed to execute command");     }
      exit(1);
}
```

- Hàm **strtok** chia một chuỗi thành các mã thông báo

```
#include <string.h>
char *strtok(char *restrict s1, const char *restrict s2);
```

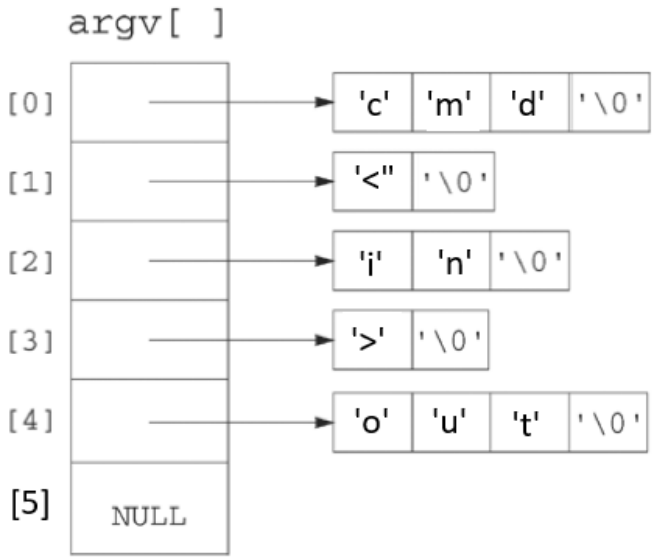
C Programming

❑ TIẾN TRÌNH ▪ Định hướng lại tiến trình con




```
int makeargv(const char *s, const char *delimiters, char ***argvp)
{  int error;
   int i;   int numtokens;   const char *snew;   char *t;
   if ((s == NULL) || (delimiters == NULL) || (argvp == NULL))   {
         errno = EINVAL;       return -1;     }
  *argvp = NULL;
   snew = s + strspn(s, delimiters);                    /* snew is real start of string */
   if ((t = malloc(strlen(snew) + 1)) == NULL)
         return -1;
   strcpy(t, snew);     numtokens = 0;
   if (strtok(t, delimiters) != NULL)                   /* count the number of tokens in s */
       for (numtokens = 1; strtok(NULL, delimiters) != NULL; numtokens++) ;
   if ((*argvp = malloc((numtokens + 1)*sizeof(char *))) == NULL)     {
         error = errno;   free(t);
         errno = error;   return -1;      }          /* insert pointers to tokens into the argument array */
    if (numtokens == 0) free(t);
    else     {
              strcpy(t, snew);          **argvp = strtok(t, delimiters);
              for (i = 1; i < numtokens; i++)      *((*argvp) + i) = strtok(NULL, delimiters);      }
     *((*argvp) + numtokens) = NULL;   return numtokens;
}
```

C Programming



- **TIẾN TRÌNH**
  - **Định hướng lại tiến trình**

```
#include <errno.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <sys/stat.h>
#define FFLAG (O_WRONLY | O_CREAT | O_TRUNC)
#define FMODE (S_IRUSR | S_IWUSR)
int parseandredirectin(char *cmd)
{       int error; int infd;                    /* redirect standard input if '<' */
        char *infile;
        if ((infile = strchr(cmd, '<')) == NULL)
                return 0;
        *infile = 0;                            /* take everything after '<' out of cmd */
         infile = strtok(infile + 1, " \t");
         if (infile == NULL)
                return 0;
         if ((infd = open(infile, O_RDONLY)) == -1)
                return -1;
         if (dup2(infd, STDIN_FILENO) == -1)
         {      error = errno;                  /* make sure errno is correct */
                close(infd);
                errno = error;
                return -1;      }
         return close(infd);
}
```

❑ SIGNAL

- *Signal* là phương thức giao tiếp giữa tiến trình này với tiến trình khác hoặc giữa kernel và process hay coi là phần mềm ngắt và là một phương tiện thực thi quy trình gián đoạn luồng lệnh.

- Có một số nguyên nhân

i) Terminal phát ra báo hiệu khi 1 phím được bấm (reset, ^D/C..)

ii) Phần cứng tạo báo hiệu: lỗi bộ nhớ, chia 0 (CRC)

iii) Hàm kill(2) cho phép một tiến trình gửi báo hiệu đến tiến trình/nhóm tiến trình khác

iv) Hàm kill(1) cho phép ta gửi báo hiệu đến tiến trình (thường chấm dứt tiến trình nền)

v) Điều kiện phần mềm tạo báo hiệu khi một tiến trình cần được báo hiệu khi các sự kiện thay đổi

- Có thể yêu cầu kernel i) tảng lờ báo hiệu ii) bắt báo hiệu – gọi hàm xem xét điều kiện báo hiệu và iii) Để báo hiệu hành động mặc định

❑ QUẢN LÍ SIGNAL

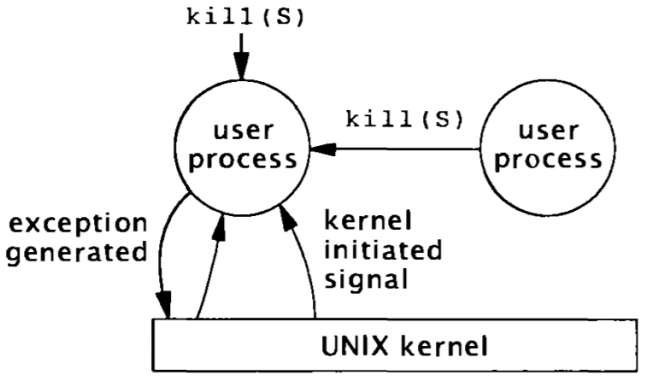
- Hàm kill() gửi báo hiệu tới tiến trình hay nhóm tiến trình

Khuôn dạng: *#include <sys/signal.h>*

int kill(pid_t pid, int signol) ;

int raise(int signo);

Both return: 0 if OK, −1 on error



- Có 4 điều kiện
  - + pid > 0: báo hiệu gửi tới tiến trình pid
  - + pid = 0: báo hiệu gửi tới nhóm tiến trình có cùng gid
  - + pid < 0: báo hiệu gửi tới nhóm tiến trình có gid = |pid|
  - + pid = -1: báo hiệu gửi tới ∀ tiến trình trong hệ thống
- ***Sys/signal*** có tới 32 *signo* có tác dụng
  - + SIGKILL : báo hiệu này làm tiến trình sẽ chấm dứt
  - + SIGSYS : báo hiệu này đưa ra khi tiến trình gọi tiến trình hệ thống với số không hợp lệ
  - + SIGPIPER: Báo hiệu đưa ra khi ghi vào pipe hay socket mà thu đã chấm dứt
  - + SIGINT: Báo hiệu đưa ra khi bấm phím ngắt như “delete” hay “^C”
  - + SIGILL: Báo hiệu đưa ra khi một tiế trình vận hành lênh không hợp lệ

……

❑ QUẢN LÍ SIGNAL

```
$ wd &
[1]     1452
$ kill -9 1452
SIGINT post by uid(13421), pid(1452)
[1] + Done     wd
```

- ***Sys/signal***

+ SIGHUP: Báo hiệu đưa ra khi kết thúc chương trình
+ SIGABORT: Báo hiệu dừng tiến trình
+ SIGALRM: Báo hiệu đưa ra bởi hàm alam
+ SIGTERM : Báo hiệu yêu cầu tiến trình kết thúc (thường hệ thống gửi shutdown
+ SIGCHLD: Báo hiệu sínhra khi tiến trình con kết thúc
+ SIGTSTP: Đầu cuối dừng báo hiệu bởi tổ hợp CTRL+Z
+ SIGQUIT: Báo hiệu đầu cuối thoát (Control-backslash)
+ SIGSEGV: Báo hiệu tham chiếu bộ nhớ không hợp lệ
+ SIGCONT: Báo hiệu tiếp tục chạy (nếu bị dừng)
+ SIGSTOP: Báo hiệu ngừng chạy
+ SIGTTIN: Báo hiệu tiến trình nền đang cố gắng đọc
+ SIGTTOU: Báo hiệu tiến trình nền đang cố gắng ghi
+ SIGUSR1/2: Báo hiệu dự phòng cho các ứng dụng người dùng

.....

❑ QUẢN LÍ SIGNAL

```
#include "apue.h"
#include <errno.h>
static void  sig_hup(int signo)
{   printf("SIGHUP received, pid = %ld\n", (long)getpid());  }
static void  pr_ids(char *name)
{   printf("%s: pid = %ld, ppid = %ld, pgrp = %ld, tpgrp = %ld\n",
     name, (long)getpid(), (long)getppid(), (long)getpgrp(), (long)tcgetpgrp(STDIN_FILENO));
    fflush(stdout);    }
int  main(void)
{       char c;   pid_t pid;
        pr_ids("parent");
        if ((pid = fork()) < 0) {
              err_sys("fork error");
        } else if (pid > 0) {                    /* parent */
              sleep(5);                          /* sleep to let child stop itself */
        } else {                                 /* child */
              pr_ids("child");
              signal(SIGHUP, sig_hup);           /* establish signal handler */
              kill(getpid(), SIGTSTP);           /* stop ourself */
              pr_ids("child");                   /* prints only if we're continued */
              if (read(STDIN_FILENO, &c, 1) != 1)
                    printf("read error %d on controlling TTY\n", errno);
        }
        exit(0);
}
```

```
$ ./a.out
parent: pid = 6099, ppid = 2837, pgrp = 6099, tpgrp = 6099
child: pid = 6100, ppid = 6099, pgrp = 6099, tpgrp = 6099
$ SIGHUP received, pid = 6100
child: pid = 6100, ppid = 1, pgrp = 6099, tpgrp = 2837
read error 5 on controlling TTY
```

❑ QUẢN LÍ SIGNAL

- Hàm ***signal(S)*** một kỹ thuật được sử dụng để thông báo tiến trình rằng một điều kiện () đã xảy ra mà tiến trình có 3 lựa chọn

+ Tảng lờ báo hiệu

+ Thực hiện nhiệm vụ mặc định

+ Cấp 1 hàm của riêng () khi có báo hiệu

- Khuôn mẫu ***signal(S)*** *#include <signal.h>*

*void (*signal (int sig, void (*func) (int))) (int);*

- SIG_ERR, SIG_DFL và SIG_IGN được định nghĩa trong <signal.h>

```
#define  _SIG_FTN (void(*) (int))
#defined SIG_HOLD()                 /* Request signal that will be help *?
#define SIG_ERR()                   /* Return value from signal in case of errot*?
#define SIG_DFL ( _SIG_FTN_0)       /* Request for default signal handling*/
#define SIG_IGN (_SIG_FTN_1)        /* Request that signal ignored */
```

C Programming

❑ QUẢN LÍ SIGNAL

- Ví dụ: Hàm ***alarm()*** dùng để gửi báo hiệu SIGALRM sau () giây kể từ khi có lời gọi đến hàm này

Khuôn dạng:

<unistd.h>

Unsigned int ***alarm****(unsigned int seconds);*

```
#include "apue.h"
#include <pwd.h>
static void  my_alarm(int signo)
{    struct passwd *rootptr;
     printf("in signal handler\n");
     if ((rootptr = getpwnam(“tuannq")) == NULL)
               err_sys("getpwnam(root) error");
     alarm(1);   }
Int  main(void)
{     struct passwd *ptr;
      signal(SIGALRM, my_alarm);
      alarm(1);
      for ( ; ; ) {
            if ((ptr = getpwnam(“tuannq")) == NULL)
                  err_sys("getpwnam error");
            if (strcmp(ptr->pw_name, “tuannq") != 0)
                  printf("return value corrupted!, pw_name = %s\n", ptr->pw_name);
       }
}
```

❑ QUẢN LÍ SIGNAL

- Hàm alarm() cho phép gửi báo hiệu SIGALRM và nêu sử dụng hàm settimer() sẽ cho phép định kì gửi . Khuôn dạng

```
#include <sys/time.h>
int getitimer(int which, struct itimerval *value);
int setitimer(int which,conststruct itimer-val *value, struct itimerval *ovalue);
```

```
struct itimerval {
    struct timeval it_interval;  /* next */
    struct timeval it_value;  };  /* current*/
struct timeval {
     long tv_sec;      /* seconds */
     long tv_usec; }; /* microseconds */
```

```
#include <sys/time.h>
#include <signal.h>
#include <stdio.h>
void handler(int sig)
{  printf("Signal handler! - got signal %d\n", sig); }
int main(int argc, char *argv[])
{    struct itimerval rttimer;
     struct itimerval old_rttimer;
     printf("setitimer demonstration!\n");
    signal(SIGALRM,handler);
     rttimer.it_value.tv_sec = 3; //seconds
     rttimer.it_value.tv_usec = 0;
     rttimer.it_interval.tv_sec = 3; //seconds
     rttimer.it_interval.tv_usec = 0;
```

```
setitimer (ITIMER_REAL, &rttimer, &old_rttimer);
while(1)
  {
     printf(".");
     sleep(1);
  }
}
```

❑ QUẢN LÍ SIGNAL

- Ví dụ: ***$cat p.c***

```
#include  <signal.h>
void inthdlr(int sig)
   {     printf("caught SIGINT (%d)\n", sig) ;
         sleep(1) ;
   }
void main()
   {     printf ( "PID=%d\n", getpid ());
         signal(SIGUSR1, SIG_IGN) ;
         signal(SIGUSR2, SIG_DFL);
         signal(SIGINT, inthdlr) ;
         while (1)
             pause ( ) ;
    }
```

int pause()

Tiến trình pause ngủ vô thời hạn cho đến khi có báo hiệu. Khi một báo hiệu được gọi, lệnh pause() trả về giá trị -1 và errno được đặt thành EINTR.

Ví dụ báo hiệu SIGUSRl, SIGUSR2 và SIGINT.

- *Với báo hiệu SIGUSR1, kernel được hướng dẫn bỏ qua báo hiệu, tức là không có hành động nào để thực hiện.*
- *Đối với tín hiệu SIGUSR2, hành vi mặc định sẽ được thực hiện. Nếu một tín hiệu SIGINT được đăng, kernel được hướng dẫn để gọi hàm inthdlr() của người dùng.*

```
$ a.out
PID=660 ,                          $ ki11 –s   2 660    #SIGINT
caught SIGINT(2)                   $ ki11 –s  16 660    #SIGUSRl
User signal 2----                  $ ki11 -s 17 660     #SIGUSR2
$
```

❑ QUẢN LÍ SIGNAL

- Ví dụ: ***$cat catches.c***

Chương trình chỉ ra một thủ tục xử lý tín hiệu đơn giản bắt được một trong hai tín hiệu do người dùng xác định và in ra số tín hiệu đó

```
$ ./a.out &
   [1] 7216
$ kill -USR1
   7216
$ kill -USR2
   7216
$ kill 7216
[1]+ Terminated ./a.out
```

```
#include "apue.h"
static void sig_usr(int); /* one handler for both signals */
Int main(void)
{
      if (signal(SIGUSR1, sig_usr) == SIG_ERR)
            err_sys("can't catch SIGUSR1");
      if (signal(SIGUSR2, sig_usr) == SIG_ERR)
            err_sys("can't catch SIGUSR2");
for ( ; ; )
        pause();
}
static void  sig_usr(int signo) /* argument is signal number */
{
      if (signo == SIGUSR1)
            printf("received SIGUSR1\n");
      else if (signo == SIGUSR2)
            printf("received SIGUSR2\n");
      else
            err_dump("received signal %d\n", signo);
      }
```

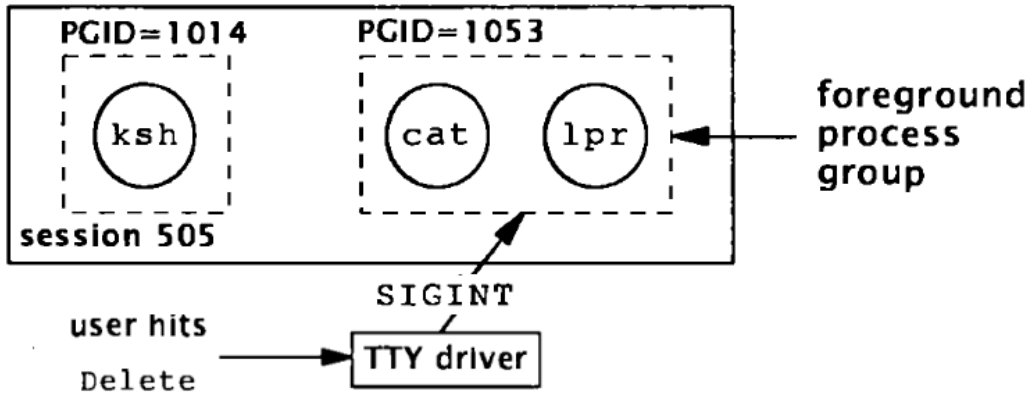
❑ QUẢN LÍ CUỘC GỌI & NHÓM TIẾN TRÌNH

- Mỗi tiến trình có PID và được trả về từ fork(S). PGID nhận dạng nhóm tiến trình gồm một hoặc nhiều tiến trình và tồn tại nếu ít nhất một tiến trình trong nhóm.
- Tiến trình cũng là thành viên của một phiên là tập hợp một hoặc nhiều tiến trình nhóm. Phiên login() là thuật ngữ để mô tả tập các sự kiện giữa login và logout.
- Một tiến trình có thể xác định PGID của chính nó hoặc của một tiến khác tồn tại trong cùng một phiên bằng lệnh getpgrp (S) và getpgid (S) tương ứng.

```
#include <sys/types>
#include <unistd.h>
        pid_t getpgrp(void) ;
        pid_t getpgid(pid_t pid) ;

#include <sys/types>
        int setpgid(pid_t pid, pid_t pgid);

#include <unistd.h>
        pid_t getsid(pid_t pid)
        pid_t setsid(void) ;
```



❑ QUẢN LÍ CUỘC GỌI & NHÓM TIẾN TRÌNH

- Đối số pid được truyền vào getpgid(s) để lấy số tiến trình nhóm. Nếu tiến trình được chỉ định không cùng phiên với tiến trình gọi getpgid (S), -1 được trả về và errno được đặt thành EPERM.

```
#include <sys/types>
#include <unistd.h>
main(int argc, char *argv[])
{       printf("process group ID for %s = %d\n", argv[O]. getpgrp()) ;
        printf("parent group ID= %d\n", getpgid(getppid())) ;
}
pid_t waitpid(pid_t pid,int*status,int options);
```

```
$ a.out
process group ID for pgrps 251
parent group ID = 250
```

C Programming

```
#include "apue.h"
#include <fcntl.h>
#include <errno.h>
void  sigint(int signo)
{ }
Int  main(void)
{  pid_t pid1, pid2;
   int fd;
   setbuf(stdout, NULL);
   signal_intr(SIGINT, sigint);
   if ((fd = open("lockfile", O_RDWR|O_CREAT, 0666)) < 0)
       err_sys("can't open/create lockfile");
    if ((pid1 = fork()) < 0) {
       err_sys("fork failed");
    } else if (pid1 == 0) {                /* child */
       if (lock_reg(fd, F_SETLK, F_RDLCK, 0, SEEK_SET, 0) < 0)
           err_sys("child 1: can't read-lock file");
       printf("child 1: obtained read lock on file\n");
       pause();
       printf("child 1: exit after pause\n");
       exit(0);
    } else {                               /* parent */
       sleep(2);
    }   if ((pid2 = fork()) < 0) {
       err_sys("fork failed");
    } else if (pid2 == 0) {                /* child */
      if (lock_reg(fd, F_SETLK, F_RDLCK, 0, SEEK_SET, 0) < 0)
          err_sys("child 2: can't read-lock file");
       printf("child 2: obtained read lock on file\n");
       pause();
       printf("child 2: exit after pause\n");
       exit(0);
    } else {                               /* parent */
       sleep(2);     }
     if (lock_reg(fd, F_SETLK, F_RDLCK, 0, SEEK_SET, 0) < 0)
         printf("parent: can't set read lock: %s\n",
       strerror(errno));
       else
        printf("parent: obtained additional read lock while"
             " write lock is pending\n");
        printf("killing child 1...\n");
        kill(pid1, SIGINT);
        printf("killing child 2...\n");
        kill(pid2, SIGINT);
       exit(0);
}
```

❑ **QUẢN LÍ TIẾN TRÌNH**

Thanks