System Programming

5. Process Control

Seung-Ho Lim

Dept. of Computer & Electronic Systems Eng.

Process ID

Getting Process IDs

```
#include <sys/types.h>
#include <unistd.h>

pid_t getpid(void);   // return PID of caller process
pid_t getpgrp(void);  // return Process GID of caller
```

- Session
  - A group of processes.
  - A process group is a kind of work, and a session is a kind of work space.
  - There is a group leader called session leader

- Process GID(group id)
  - will be a session leader's pid (usually, shell's pid)

Getting Parent Process ID

```
#include <sys/types.h>
#include <unistd.h>

pid_t getppid(void);   // return PID of caller's parent
```

```
pid_t getpgid(pid_t tpid);
```

- parameters
  - tpid : process id for inquiry
- return
  - Inquired process' group ID if Ok
  - -1 on error

PID example (1)

pid-ex.c

```
#include <sys/types.h>
#include <unistd.h>
#include <stdio.h>

int main()
{
   int pid;
   printf("Original process : pid = %d, ppid = %d, pgrp
   = %d, pgid = %d\n", getpid(), getppid(), getpgrp(),
   getpgid(getpid()));

   pid = fork();
   if(pid != 0) {
        printf("Parent process : pid = %d, ppid = %d, pgrp
   = %d, pgid = %d\n", getpid(), getppid(), getpgrp(),
   getpgid(getpid()));
}
```

PID example (2)

```
    else {
        printf("Child process : pid = %d, ppid = %d, pgrp
    = %d, pgid = %d\n", getpid(), getppid(), getpgrp(),
    getpgid(getpid()));
    }
    return 0;
}
```

- *Run & Result*

```
$ ./a.out
Original process : pid = 26392, ppid = 25089, pgrp = 26392, pgid = 26392
Parent process : pid = 26392, ppid = 25089, pgrp = 26392, pgid = 26392
Child process : pid = 26393, ppid = 26392, pgrp = 26392, pgid = 26392

$
```

Process Group Leader

- A session leader process' pid will be the *pgid.*
  - When we log on remote machine, the shell will be session leader.
  - A signal can be sent to the whole group of processes.
  - If the session leader exits
    - SIGHUP signal is delivered to all process in the process group
    - all processes of the group will be terminated
  - Thus, when we log out the machine, the shell and all descendant processes will be terminated.
- To continue the execution of a child process after logout
  - use **nohup** command with background mode
  - instead of standard I/O, use file I/O

```
$ nohup command .... &
```

Setting Process Group ID

```
#include <sys/types.h>
#include <unistd.h>

int setpgid(pid_t pid, pid_t pgid);
```

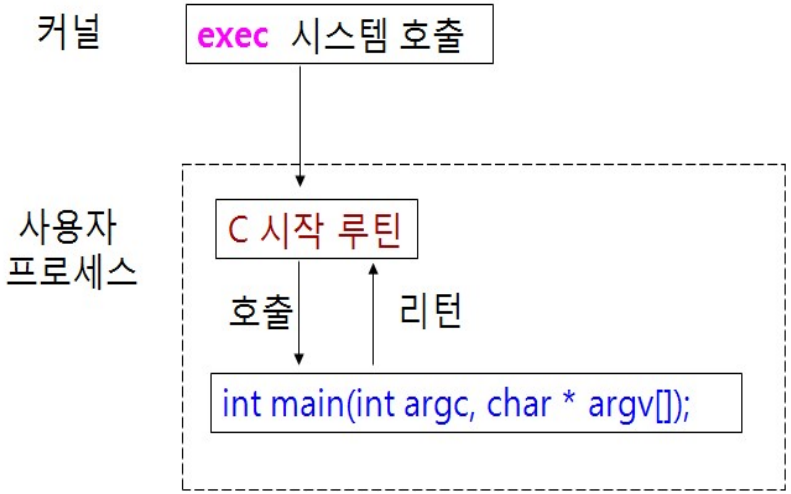
- parameters
  - *pid*: process ID
  - *pgid*: process group ID
- return
  - 0 if Ok
  - -1 on error

pid	*pgid*	*description*
0	0	Call process becomes the group leader
0	x	Set the calling process's pgid as x
x	0	Set the calling process's pgid as the pgid of the process x
x	x	Process with pid x will becomes the group leader
x	y	Set the process(pid = x)'s pgid as y

Process Start & Exit

Process Running Start

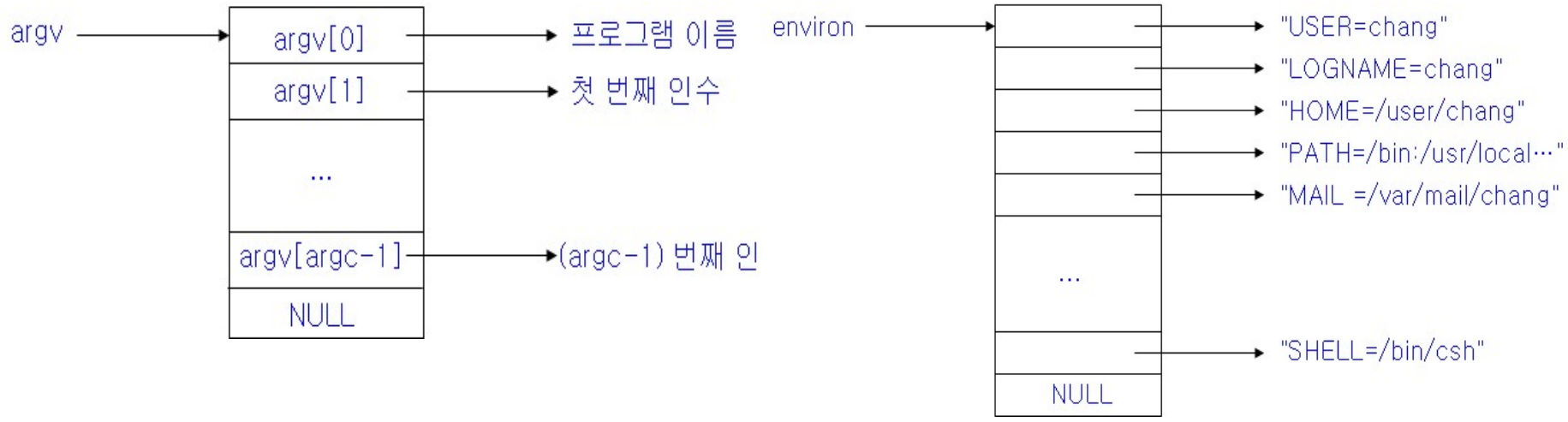
- exec system call
  - Pass command line arguments and environment variables to the C startup routine
  - Run the program.
- C start-up routine
  - Passing command-line arguments and environment variables while calling the main function



Process Running Start

- Command line arguments / environment variables

```
int main(int argc, char *argv[]);
argc : 명령줄 인수의 개수
argv[] : 명령줄 인수 리스트를 나타내는 포인터 배열
```



Environment

```
#include <stdlib.h>

char *getenv(const char *name);
```

- parameters
  - name: name of environment
- return
  - Returns the value of the environment variable name.
  - If the variable does not exist, NULL is returned.

Environment

```
#include <stdlib.h>

int putenv(const char *name);
int setenv(const char *name, const char *value, int
rewrite);
```

- parameters
  - name: name of environment
  - value : values of set for the name
  - rewrite : If name already exists, if the rewrite value is not 0, the original value is replaced with the new value. If the rewrite value is 0, it is left as it is.
- return
  - 0 if Ok
  - -1 on error

Environment example

env-ex.c

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char *argv[])
{
   char **ptr;
   char *nptr;
   extern char **environ;

   for (ptr = environ; *ptr != 0; ptr++) /* print all environments */
      printf("%s \n", *ptr);

    nptr = getenv("HOME");
    printf("HOME = %s \n", nptr);

    nptr = getenv("SHELL");
    printf("SHELL = %s \n", nptr);

    nptr = getenv("PATH");
    printf("PATH = %s \n", nptr);

   exit(0);
}
```

Process Termination

- Normal termination
  - When main () finishes execution and returns, the C startup routine calls exit () with this return value.
  - Call exit () directly from within the program
  - Call _exit () directly from within the program

- Abnormal termination
  - abort ()
  - Sending a SIGABRT signal to the process, terminating the process abnormally
  - Termination by signal

Process Termination

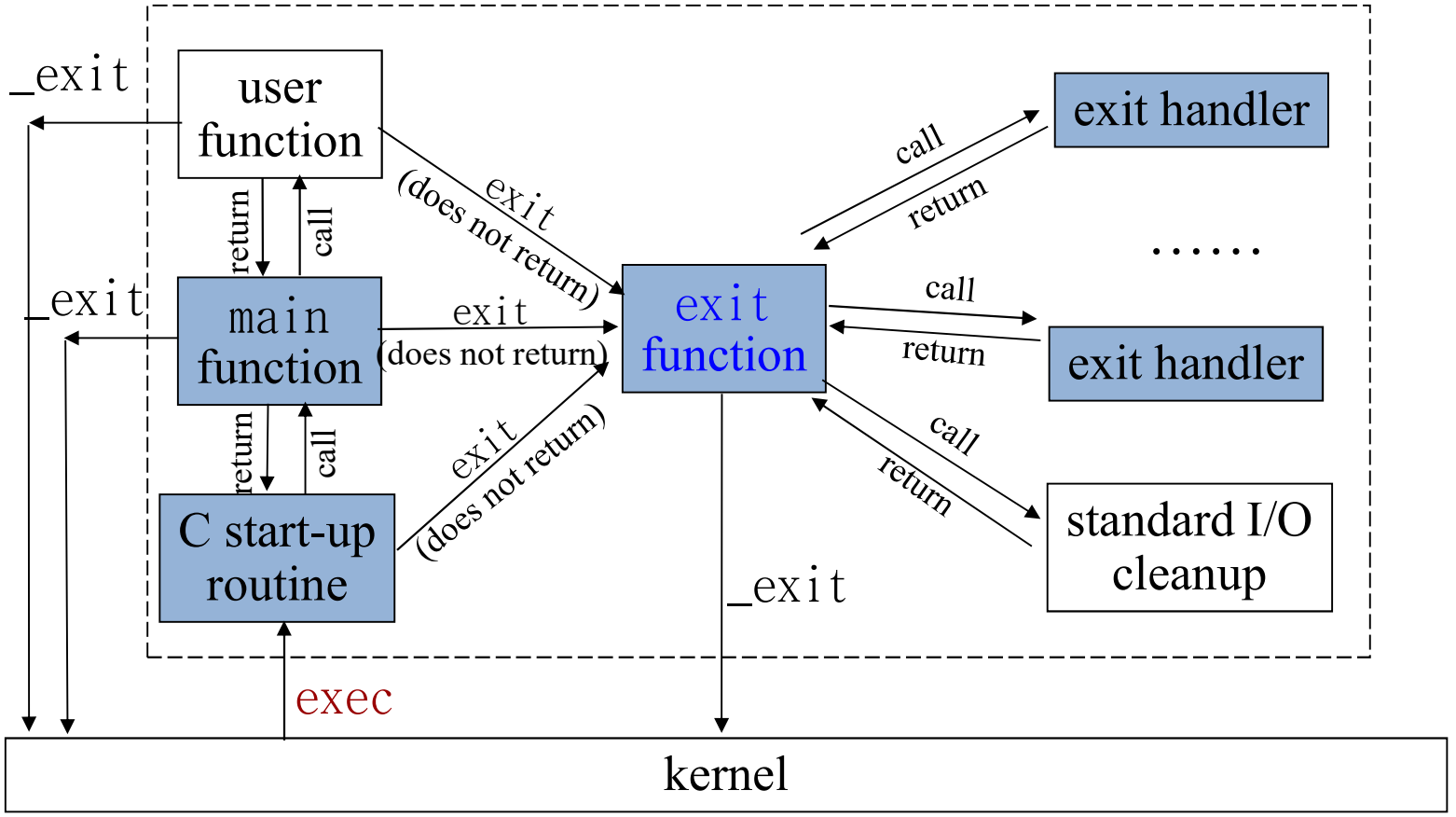
- exit ()
  - Closes all open streams (fclose) and writes the contents of the output buffer to disk (fflush).
  - Pass the exit code to the parent process.

```
#include <stdlib.h>
void exit(int status);
//After rearranging, the process ends normally.
```

  - _exit ()

```
#include <stdlib.h>
void _exit(int status);
//Immediately terminate the process without
rearranging.
```

Program Start & Termination



Exit handler: *atexit()*

```
#include <stdlib.h>

int  atexit(void (*func)(void));
```

- registers exit handler(s) which will be called at exit time
  - in the case that multiple functions were registered, they will be called in the *reverse order*.
- parameters
  - *func*: function which will be called at the *exit* time
  - this function is called *exit handler*
- return
  - 0 if Ok
  - nonzero on error

Function pointer example ()

fptr-ex.c

```
#include<stdio.h>
void greet();
void caller(void (*myFunction)(void));

int main()
{
          void (*sayHello)(void);

          printf("Calling greet() directly.\n");
          greet();

          printf("\n");

          printf("Calling greet() via a pointer.\n");
          sayHello = greet;
          sayHello();
          printf("\n");

          printf("Calling greet() via another function.\n");
          caller(greet);
}

void greet(){
      printf("Hello, world!\n");
}

void caller(void (*myFunction)(void)){
      myFunction();
}
```

Function pointer example (2)

fptr-ex.c

```
$./fptr-ex
Calling greet() directly.
Hello, world!

Calling greet() via a pointer.
Hello, world!

Calling greet() via another function.
Hello, world!
```

atexit example (1)

atexit-ex.c

```
#include <stdio.h>
#include <stdlib.h>

#define TMPFILE  "/tmp/mylog"

static void myexit(void);

int main(void)
{
        FILE *fp;

        if (atexit(myexit) != 0) {
                perror("atexit error");
                exit(1);
        }
```

atexit example (2)

atexit-ex.c

```
        if ((fp = fopen(TMPFILE, "a+")) == NULL) {
                perror("fopen error");
                exit(2);
        }
        fprintf (fp, "This is temporary log entry of pid %d\n",
                        getpid());
        close (fp);
        exit(0);
}

static void myexit(void)
{
        if(unlink(TMPFILE)) {
             perror("myexit : unlink");
             exit(3);
        }
}
```

Daemons

- Daemon
  - Background process not connected to terminal
  - It is usually started from root at boot time, or from some special user.
  - Daemons have d at the end of their names.
    - crond, sshd,
- Daemon two conditions
  - What should be executed as a child of init
  - What should not be connected to the terminal

Daemons

- To become a daemon, the following steps are performed.
  1. fork() call. Creating a new process
  2. Call exit() in parent. Make sure that the daemon's parents do not exist.
  3. call setsid(). New process groups and sessions are assigned.
  4. Using chdir() to go to the lot directory
  5. Close all file descriptors.
  6. Connect file descriptor 0,1,2 (standard input, standard output, standard error) to / dev / null

setsid()

```
#include <sys/types.h>
#include <unistd.h>

pid_t setsid(void);   // create new session
```

- Create session
  - Create a new session and become the leader of the session.
  - Create a new process group and become the leader of that process group.
  - You cannot use the control terminal of the previous session. It does not have a control terminal.

- will be a session leader's pid (usually, shell's pid)

Daemon example

daemon-ex.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    pid_t   pid;
    int i = 10000;

    if ((pid = fork()) < 0) {
        perror("fork error");
    } else if (pid == 0) {  /* child process → daemon */
        printf("Child: pid=%d\n",getpid());
        close(0);
        close(1);
        close(2);
        setsid(); //terminal close, create new session and process group
        while(1) {
                printf("c(%d)\n",i);
                i++;
                sleep(1);
        }
```

Daemon example

daemon-ex.c

```
    } else {      /* parent process */
        printf("Parent: pid=%d\n",getpid());
        printf("Parint: Child pid=%d\n",pid);

        sleep(1);

        printf("Parent : exit\n");

        exit(0);
    }
}
```

- Run & Result

```
$./daemon-ex
Parent: pid=7183
Parint: Child pid=7184
Child: pid=7184
Parent : exit
$ ps -ef
shlim     7184     1  0 00:56 ?        00:00:00 ./daemon-ex
```

Zombie example

zombie-ex.c

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>

int main(void)
{
    pid_t   pid;
    int i = 10000;

    if ((pid = fork()) < 0) {
        perror("fork error");
    } else if (pid == 0) {  /* child process → daemon */
        printf("Child: pid=%d\n",getpid());

        while(1) {
                printf("c(%d)\n",i);
                i++;
                sleep(1);
        }
```

Zombie example

zombie-ex.c

```
    } else {      /* parent process */
        printf("Parent: pid=%d\n",getpid());
        printf("Parint: Child pid=%d\n",pid);

        sleep(1);

        printf("Parent : exit\n");

        exit(0);
    }
}
```

- Run & Result

```
$./zombie-ex
Parent: pid=7199
Parint: Child pid=7200
Child: pid=7200
c(10000)
Parent : exit
c(10001)
c(10002)
…
```

system()

```
#include <stdlib.h>

int  system(const char *string);
```

- execute a shell command given in the string

- parameters
  - *string*: shell command string including options
  - e.g. ls -al, ps, kill -9, ...
- return
  - shell's exit code if Ok
  - -1 on error

system() example ()

system-ex.c

```
#include <stdio.h>

main(int ac, char *av[])
{
    int i;
    char cmdstr[1024];

    strcpy(cmdstr, "/bin/ls ");

    for(i=1;i < ac;i++) {
        strcat(cmdstr, av[i]);
        strcat(cmdstr, " ");
    }
    fprintf(stdout, "cmdstr = \"%s\"\n", cmdstr);

    system(cmdstr);

    exit(0);
}
```

- Run & Result

```
$./system-ex
cmdstr = "/bin/ls "
echoall.c  fork.c  fwait.c  nexec.c  race.c  system-ex.c
```

Process time

```
#include <sys/times.h>

clock_t times(struct tms *buf);
        Returns:
struct tms (
  clock_t tms_utime; /* user cpu time */
  clock_t tms_stime; /* system cpu time */
  clock_t tms_cutime; /* child user cpu time */
  clock_t tms_cstime; /* child system cpu time */
}
```

- return
  - elapsed wall clock time in clock ticks if OK
  - -1 on error
- times
  - *wall clock time*: the amount of time the process takes to run and depends on the system loads.
  - *user CPU time*: attributed to user instructions
  - *system CPU time*: attributed to the kernel (on behalf of the user process)

Process time example (1)

time-ex.c

```
#include <sys/times.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.h>

static void pr_times(clock_t, struct tms *, struct tms *);
static void do_cmd(char *);

void err_sys(char *p) { perror(p); exit(-1); }

int main(int argc, char *argv[])
{
    int     i;

    for (i = 1; i < argc; i++)
        do_cmd(argv[i]);    /* once for each command-line arg */
    exit(0);
}
```

Process time example (2)

time-ex.c

```
static void do_cmd(char *cmd)       /* execute and time the "cmd" */
{
    struct tms  tmsstart, tmsend;
    clock_t     start, end;
    int         status;

    printf("\ncommand: %s\n", cmd);

    if ((start = times(&tmsstart)) == -1)   /* starting values */
        err_sys("times error");

    if ((status = system(cmd)) < 0)     /* execute command */
        err_sys("system() error");

    if ((end = times(&tmsend)) == -1)       /* ending values */
        err_sys("times error");

    pr_times(end-start, &tmsstart, &tmsend);
}
```

Process time example (3)

time-ex.c

```
static void pr_times(clock_t real, struct tms *tmsstart, struct tms *tmsend)
{
    static long     clktck = 0;

    if (clktck == 0)    /* fetch clock ticks per second first time */
        if ((clktck = sysconf(_SC_CLK_TCK)) < 0)
            err_sys("sysconf error");
    printf("  real:  %7.2f\n", real / (double) clktck);
    printf("  user:  %7.2f\n",
            (tmsend->tms_utime - tmsstart->tms_utime) / (double) clktck);
    printf("  sys:   %7.2f\n",
            (tmsend->tms_stime - tmsstart->tms_stime) / (double) clktck);
    printf("  child user:  %7.2f\n",
            (tmsend->tms_cutime - tmsstart->tms_cutime) / (double) clktck);
    printf("  child sys:   %7.2f\n",
            (tmsend->tms_cstime - tmsstart->tms_cstime) / (double) clktck);
}
```

Process time example (4)

sleep5.c

```
#include <unistd.h>

int main(int argc, char **argv)
{
     sleep(5);
}
```

calc.c

```
#include <stdio.h>

int main()
{
    double pi=0,temp=1,p=-1,num=1;

    while(num<1000000000)
    {
        p*=-1;
        pi+=p*1.0/temp;
        temp+=2;
        num++;
    }
}
```

Process time example (3)

- *Run & Result*

```
$./time-ex ./sleep5 ./calc

command: ./sleep5
  real:     5.00
  user:     0.00
  sys:      0.00
  child user:     0.00
  child sys:      0.00

command: ./calc
  real:     6.07
  user:     0.00
  sys:      0.00
  child user:     6.05
  child sys:      0.00
```

Process time example (5)

- *Run & Result*

```
$./time-ex ./sleep5 ./calc

command: ./sleep5
  real:     5.00
  user:     0.00
  sys:      0.00
  child user:     0.00
  child sys:      0.00

command: ./calc
  real:     6.07
  user:     0.00
  sys:      0.00
  child user:     6.05
  child sys:      0.00
```

HW 3

- Write source code and test results
  - daemon-ex.c
  - zombie-ex.c
  - time-ex.c with sleep5.c, calc.c
- Write report with test results
- Daemon process & zombie process should be killed after testing
  - ps -ef
  - Kill -9 (PID)
- Due date
  - 4/23