

Process – Part2

Process Attributes

- Process ID
- Process groups and process group ID
- Environment
- Current working and root directory
- User and group ID
- Process Priorities

Process ID

- Process ID
 - Every process has a unique process ID.
 - The index of the process table entry in the kernel.
 - Often used as a piece of their identifiers, to guarantee uniqueness.
- Getting process ID
- `#include <sys/types.h>`
`#include <unistd.h>`
`pid_t getpid(void);`
`pid_t getppid(void);`

Process Group ID

- Process group ID
 - Allow processes to be usefully placed into groups
 - Typical example
 - `$ who | awk '{print $1}' | sort -u`
 - Useful when handling a set of processes as a while using an IPC mechanism called signals
 - If a process has the same ID as the process group ID, it is deemed the leader of the process group
- Getting process group ID

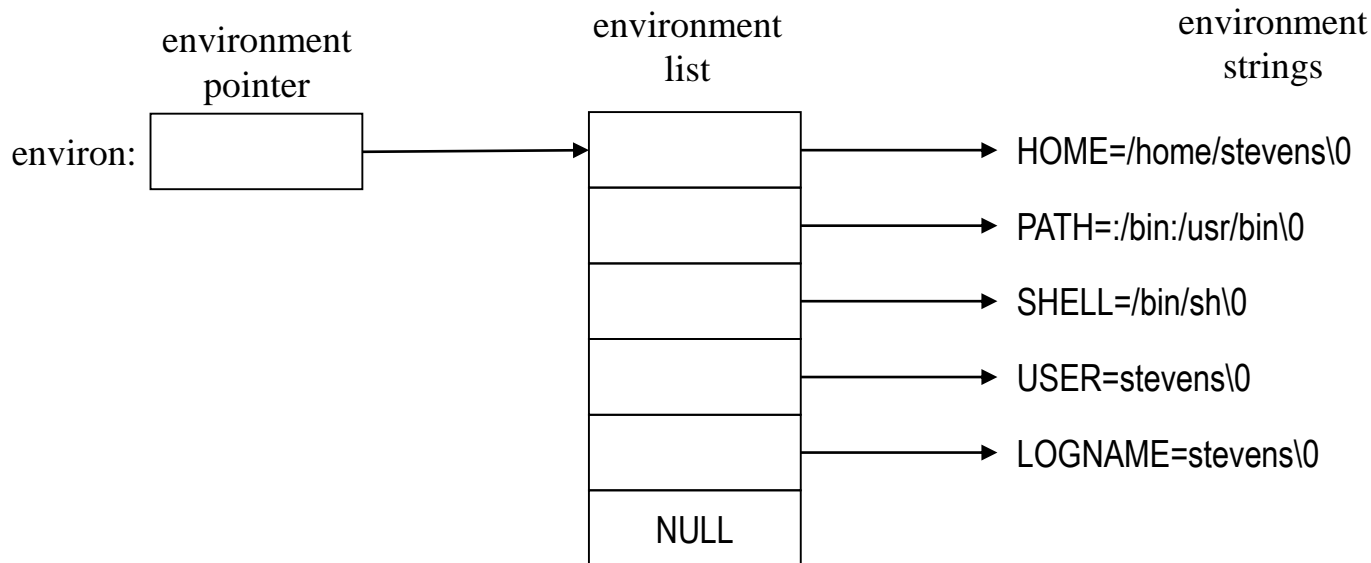
Getting/Changing Process Group ID

- `#include <sys/types.h>`
`#include <unistd.h>`
`pid_t getpgrp(void);`
 - Returns the process group ID of the current process`int setpgid(pid_t pid, pid_t pgid);`
 - Set the process group ID of the process with an ID of `pid` to `pgid`

Environment

- Process's environment is a collection of null-terminated strings as follows:
 - name = something

```
main(int argc, char **argv, char **envp)
```



Example #9: Environment List

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

```
void main(int argc, char *argv[], char
    *envp[])
{
    int i;
    extern char **environ;

    printf("from argument envp\n");

    for (i = 0; envp[i]; i++)
        puts(envp[i]);

    printf("\nFrom global variable
    environ\n");

    for (i = 0; environ[i]; i++)
        puts(environ[i]);
}
```

```
dhlee72@cse:~>a.out
from argument envp
USER=dhlee72
LOGNAME=dhlee72
HOME=/home/faculty/dhlee72
PATH=/usr/local/bin:/bin:/home/faculty/dhle
e72/bin:/home/faculty/dhlee72/javacc-
3.2/bin:/usr/bin:/bin:/usr/sbin:/sbin:/usr/etc
:/etc:/usr/X11R6/bin:/home/faculty/dhlee72
/jdk1.3.1_01/bin:/home/faculty/dhlee72/om
ni/bin/i586_linux_2.0_glibc2.1:.
MAIL=/var/mail/dhlee72
SHELL=/bin/tcsh
.....
```

getenv



- `#include <stdlib.h>`

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

- Searches the environment list for a string that matches the string pointed to by *name*.
- Returns a pointer to the value in the environment, or NULL if there is no match.

putenv



- `#include <stdlib.h>`

`int putenv(const char *string);`

- Adds or changes the value of environment variables.
- The argument *string* is of the form `name=value`.
- If `name` does not already exist in the environment, then *string* is added to the environment.
- If `name` does exist, then the value of `name` in the environment is changed to `value`.
- Returns zero on success, or -1 if an error occurs.

Example #10: getenv, putenv

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

void main(void)
{
    printf("Home directory is %s\n", getenv("HOME"));

    putenv("HOME=/");

    printf("New home directory is %s\n", getenv("HOME"));
}
```

User and Group ID

- User ID
 - Real user ID
 - Identifies the user who is responsible for the running process.
 - Effective user ID
 - Used to assign ownership of newly created files, to check file access permissions, and to check permission to send signals to processes.
 - To change `uid`: executes a *setuid-program* that has the `setuid` bit set or invokes the `setuid` system call.
 - `setuid(uid)` system call:
 - Typical program that calls the `setuid(uid)` system call
 - `passwd`, `login`, `mkdir`, etc.
 - Real and effective `uid`: `inherit` (`fork`), `maintain` (`exec`).
- Group ID
 - Real, effective

Read IDs

- `pid_t getuid(void);`
 - Returns the real user ID of the current process
- `pid_t geteuid(void);`
 - Returns the effective user ID of the current process
- `gid_t getgid(void);`
 - Returns the real group ID of the current process
- `gid_t getegid(void);`
 - Returns the effective group ID of the current process

Change UID and GID

- `#include <unistd.h>`
`#include <sys/types.h>`
`int setuid(uid_t uid)`
 - Sets the effective user ID of the current process.
 - Superuser process resets the real and effective user IDs to *uid*.
 - Non-superuser process can set effective user ID to *uid*, only when *uid* equals real user ID
 - `int setgid(gid_t gid)`
 - cf. `setreuid()`, `setregid()`, `seteuid()`, `setegid()`

Current Root Directory

- Root directory
 - Each process is associated with a root directory used in absolute pathname searches
 - Root directory of a process is initially determined by that of its parent process
- Changing root directory
- `#include <unistd.h>`
`int chroot(const char *path)`
 - Change a root directory of a process
 - *path* points to a pathname naming a directory
 - Return 0 on success, -1 on failure

Process Priorities

- Process priorities range from 0 to a system-dependent maximum
 - The higher the number, the lower the process' priority
 - In Linux, -20 ~ +19
- Changing the priority of a process
 - nice(int *increment*)
 - Add *increment* to the process nice value
 - Users are allowed to lower the priorities of their process
 - Only superuser increase their priority by using a negative value as the nice system call
 - cf. getpriority(), setpriority()

System Function

- `#include <stdlib.h>`

`int system (const char *string);`

- Executes a command specified in *string* by calling `/bin/sh -c string`, and returns after the command has been completed.
- Implemented by calling `fork`, `exec`, and `waitpid`
- Return value
 - If either `fork` fails or `waitpid` returns error, `system` returns -1
 - If the `exec` fails, the return value is as if the shell had executed `exit(127)`.
 - Otherwise, the return value is the termination status of the shell, in format specified for `waitpid`

Example #11: system

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

#define CMDLEN 80

void main(void)
{
    char cmdstr[CMDLEN];

    printf("Enter command to run: ");
    fflush(stdout);
    gets(cmdstr);
    system(cmdstr);
}
```

More Examples #1: exit (1)

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

void pr_exit(int status)
{
    if (WIFEXITED(status))
        printf("normal termination, exit status = %d\n", WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        printf("abnormal termination, signal number = %d%s\n", WTERMSIG(status),
#ifdef WCOREDUMP
            WCOREDUMP(status) ? " (core file generate)" : " ";
#else
            " ");
#endif
    else if (WIFSTOPPED (status))
        printf("child stopped, signal number = %d\n", WSTOPSIG(status));
}
```

More Examples #1: exit (2)

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

int main(void)
{
    pid_t pid;
    int status;

    if ((pid = fork()) < 0) {
        perror("fork error\n");
        exit(1);
    }
    else if (pid == 0)    /* child */
        exit(7);
```

More Examples #1: exit (3)

```
if (wait(&status) != pid) { /* wait for child */  
    perror("wait error\n");  
    exit(1);  
}  
pr_exit(status);
```

```
if ((pid = fork()) < 0) {  
    perror("wait error\n");  
    exit(1);  
}  
else if (pid == 0) /* child */  
    abort(); /* generate SIGABRT */
```

```
if (wait(&status) != pid) { /* wait for child */  
    perror("wait error\n");  
    exit(1);  
}  
pr_exit(status); /* and print its status */
```

More Examples #1: exit (4)

```
if ((pid = fork()) < 0) {  
    perror("fork error\n");  
    exit(1);  
}  
else if (pid == 0) /* child */  
    status /= 0;    /* divide by 0 generates SIGFPE */  
  
if (wait(&status) != pid) {    /* wait for child */  
    perror("wait error");  
    exit(1);  
}  
pr_exit(status);    /* and print its status*/  
exit(0);  
}
```

More Examples #2: fork (1)

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

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

int main(void)
{
    pid_t pid;

    if ((pid = fork()) < 0)
        err_sys("fork error");
    else if (pid == 0) { /* first child */
        if ( (pid = fork()) < 0)
            err_sys("fork error");
        else if (pid > 0)
            exit(0);
```

More Examples #2: fork (2)

```
/* parent from second fork == first child */  
/* We're the second child; our parent becomes init as soon as our real parent calls  
   exit() in the statement above. Here's where we'd continue executing, knowing  
   that when we're done, init will reap our status. */
```

```
sleep(2);  
printf("second child, parent pid = %d\n", getppid());  
exit(0);  
}
```

```
if (waitpid(pid, NULL, 0) != pid)  
    /* wait for first child */  
    err_sys("waitpid error");
```

```
/* We're the parent (the original process); we continue executing, knowing that  
   we're not the parent of the second child. */
```

```
exit(0);
```

```
}
```

More Examples #3: fork

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

static void charatime (char *str)
{
    char *ptr;
    int c;

    setbuf(stdout, NULL);    /* set
unbuffered */
    for (ptr = str; c = *ptr++;)
        putc(c, stdout);
}
```

```
int main(void)
{
    pid_t pid;

    if ((pid = fork()) < 0) {
        perror("fork error\n");
        exit(1);
    }
    else if (pid == 0)
        charatime("output from
child\n");
    else
        charatime("output from
parent\n");

    exit(0);
}
```


More Examples #4: fork

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

static void charatatime(char *str)
{
    char *ptr;
    int c;

    setbuf(stdout, NULL);    /* set unbuffered
    */
    for (ptr = str; c = *ptr++; )
        putc(c, stdout);
}
```

```
int main(void)
{
    pid_t pid;

    if( (pid = fork()) < 0) {
        perror("fork error\n");
        exit(1);
    }
    else if (pid == 0) {
        pause();    /* parent goes first */
        charatatime("output from child\n");
    }
    else {
        charatatime("output from parent\n");
        kill(pid, SIGALRM);
    }
    exit(0);
}
```

More Examples #5: exec (1)

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

char *env_init[] = {"USER=unknow", "PATH=/tmp", NULL};

int main(void)
{
    pid_t pid;

    if ((pid = fork()) < 0) {
        perror("fork error\n");
        exit(1);
    }
    else if (pid == 0) {
        /* specify pathname, specify environment */
        if (execle ("/home/stevens/bin/echoall", "echoall", "myarg1", "MY AGR2", (char *) 0,
env_init) < 0) {
            fprintf(stderr, "execle error\n");
            exit(1);
        }
    }
}
```

More Examples #5: exec (2)

```
if (waitpid(pid, NULL, 0) < 0) {
    perror("wait error\n");
    exit(1);
}
else if (pid == 0) {
    /* specify filename, inherit environment */
    if (execlp("echoall", "echoall", "only 1 arg", (char *) 0) < 0) {
        fprintf(stderr, "execlp error\n");
        exit(1);
    }
}
exit(0);
}
```

More Examples #6: system (1)

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

int mysystem(const char *cmd) /* version without signal handling */
{
    pid_t pid;
    int status;

    if (cmd == NULL)
        return(1); /* always a command processor with UNIX */

    if ((pid = fork()) < 0) {
        /* probably out of processes */
        status = -1;
    }
    else if (pid == 0) {
        /* child */
        execl("/bin/sh", "sh", "-c", cmd, (char *) 0);
        _exit(127); /* execl error */
    }
}
```

More Examples #6: system (2)

```
else {
    /* parent */
    while (waitpid(pid, &status, 0) < 0)
        if (errno != EINTR) {
            status = -1; /* error other than EINTR from waitpid() */
            break;
        }
    return(status);
}
```

More Examples #6: system (3)

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

void pr_exit(int status)
{
    if (WIFEXITED(status))
        printf("normal termination, exit status = %d\n", WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        printf("abnormal termination, signal number = %d%s\n", WTERMSIG(status),
#ifdef WCOREDUMP
            WCOREDUMP(status) ? " (core file generate)" : " ";
        #else
            " ");
        #endif
    else if (WIFSTOPPED (status))
        printf("child stopped, signal number = %d\n", WSTOPSIG(status));
}
```

More Examples #6: system (4)

```
int main(void)
{
    int status;

    if ((status = mysystem("data")) < 0) {
        perror("mysystem() error\n");
        exit(1);
    }
    pr_exit(status);

    if ((status = mysystem("nosuchcommand")) < 0) {
        perror("mysystem() error\n");
        exit(1);
    }
    pr_exit(status);

    if ((status = mysystem("who; exit 44")) < 0) {
        perror("system() error\n");
        exit(1);
    }
    pr_exit(status);
    exit(0);
}
```

More Examples #7: fork (1)

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

int main(void)
{
    pid_t pid;

    if ((pid = fork()) < 0) {
        perror("fork error\n");
        exit(1);
    }
    else if (pid != 0) {
        /* parent */
        sleep(2);
        exit(2);    /* terminate with exit status 2 */
    }
}
```


More Examples #7: fork (2)

```
/* first child */
if ((pid = fork()) < 0) {
    perror("fork error\n");
    exit(1);
}
else if (pid != 0) {
    sleep(4);
    abort();    /* terminate with core dump */
}

/* second child */
if ((pid = fork()) < 0) {
    perror("fork error\n");
    exit(1);
}
else if (pid != 0) {
    execl("/usr/bin/dd", "dd", "if=/boot", "of=/dev/null", NULL);
    exit(7);    /* shouldn't get here */
}
```

More Examples #7: fork (3)

```
/* third child */
if ((pid = fork()) < 0) {
    perror("fork error\n");
    exit(1);
}
else if (pid != 0) {
    sleep(8);
    exit(0);    /* normal exit */
}

/* fourth child */
sleep(6);
kill(getpid(), SIGKILL);    /* terminate with signal, no core dump */
exit(6);    /* shouldn't get here */
}
```

More Examples #8: system (1)

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

void pr_exit(int status)
{
    if (WIFEXITED(status))
        printf("normal termination, exit status = %d\n", WEXITSTATUS(status));
    else if (WIFSIGNALED(status))
        printf("abnormal termination, signal number = %d%s\n", WTERMSIG(status),
#ifdef WCOREDUMP
            WCOREDUMP(status) ? " (core file generate)" : " ";
#else
            " ");
#endif
    else if (WIFSTOPPED (status))
        printf("child stopped, signal number = %d\n", WSTOPSIG(status));
}
```

More Examples #8: system (2)

```
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) {
            fprintf(stderr, "sysconf error\n");
            exit(1);
        }
    fprintf(stderr, " real: %7.2f\n", real / (double) clktck);
    fprintf(stderr, " user: %7.2f\n", (tmsend->tms_utime - tmsstart->tms_utime) / (double) clktck);
    fprintf(stderr, " sys: %7.2f\n", (tmsend->tms_stime - tmsstart->tms_stime) / (double) clktck);
    fprintf(stderr, " child user: %7.2f\n", (tmsend->tms_cutime - tmsstart->tms_cutime) / (double) clktck);
    fprintf(stderr, " child sys: %7.2f\n", (tmsend->tms_cstime - tmsstart->tms_cstime) / (double) clktck);
}

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

More Examples #8: system (3)

```
fprintf(stderr, "\ncommand: %s\n", cmd);

if ((start = time(&tmsstart)) == -1) { /* starting values */
    fprintf(stderr, "time error\n");
    exit(1);
}

if ((status = system(cmd)) < 0) { /* execute command */
    fprintf(stderr, "system() error\n");
    exit(1);
}

if ((end = times(&tmsend)) == -1) { /* ending values */
    fprintf(stderr, "times error\n");
    exit(1);
}

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

More Examples #8: system (4)

```
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);
}
```