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>

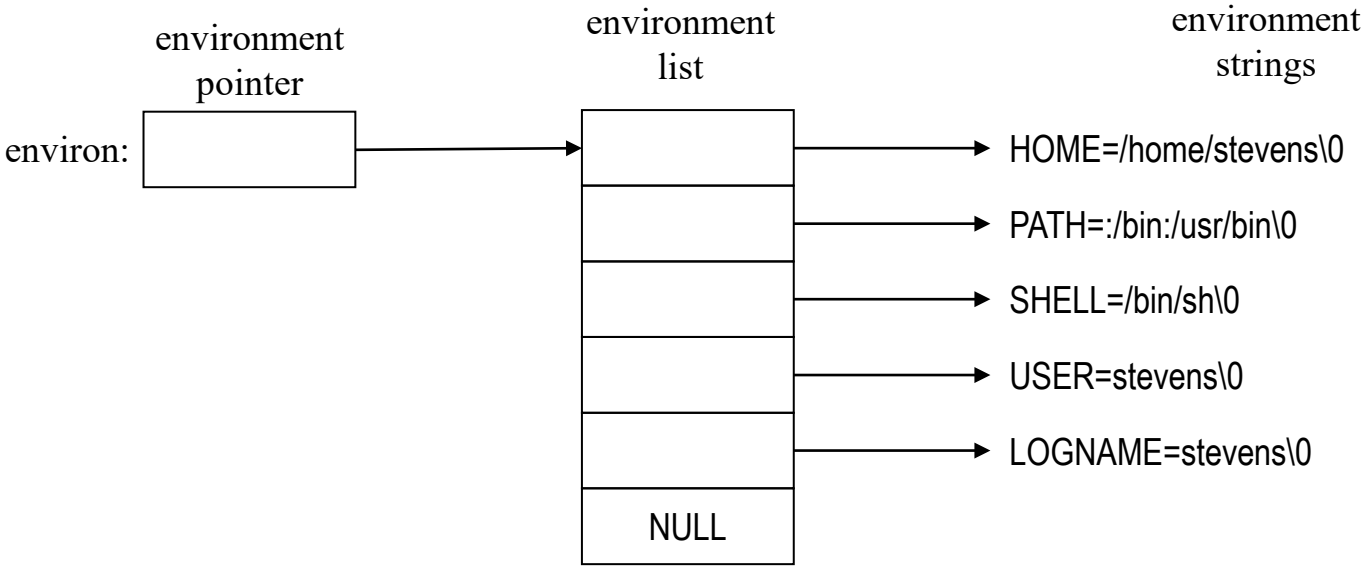
  pid_t getpgrp(void);
  - Returns the process group ID of the current process

  int setpgid(pit_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

```c
#include <stdio.h>

int 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]);

  return 0;
}
```

```
from argument envp
PATH=/home/runner/.apt/usr/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin
HOSTNAME=7acccaaa06c1
XDG_CONFIG_HOME=/config
LC_ALL=en_US.UTF-8
LANG=en_US.UTF-8
APT_OPTIONS=-o debug::nolocking=true -o dir::cache=/tmp/apt/cache -
o dir::state=/tmp/apt/state -o dir::etc::sourcelist=/tmp/apt/source
s/sources.list
LD_LIBRARY_PATH=/home/runner/.apt/usr/lib/x86_64-linux-gnu:/home/ru
nner/.apt/usr/lib/i386-linux-gnu:/home/runner/.apt/usr/lib:
LIBRARY_PATH=/home/runner/.apt/usr/lib/x86_64-linux-gnu:/home/runne
r/.apt/usr/lib/i386-linux-gnu:/home/runner/.apt/usr/lib:
INCLUDE_PATH=/home/runner/.apt/usr/include:/.apt/usr/include/x86_64
-linux-gnu:
CPATH=
CPPPATH=
PKG_CONFIG_PATH=/.apt/usr/lib/x86_64-linux-gnu/pkgconfig:/.apt/usr/
lib/i386-linux-gnu/pkgconfig:/.apt/usr/lib/pkgconfig:
LD_PRELOAD=/usr/local/lib/repl.so
DISPLAY=MAGIC
HOME=/home/runner
TERM=xterm-256color

From global variable environ
PATH=/home/runner/.apt/usr/bin:/usr/local/sbin:/usr/local/bin:/usr/
sbin:/usr/bin:/sbin:/bin
HOSTNAME=7acccaaa06c1
XDG_CONFIG_HOME=/config
LC_ALL=en_US.UTF-8
LANG=en_US.UTF-8
APT_OPTIONS=-o debug::nolocking=true -o dir::cache=/tmp/apt/cache -
o dir::state=/tmp/apt/state -o dir::etc::sourcelist=/tmp/apt/source
s/sources.list
LD_LIBRARY_PATH=/home/runner/.apt/usr/lib/x86_64-linux-gnu:/home/ru
nner/.apt/usr/lib/i386-linux-gnu:/home/runner/.apt/usr/lib:
LIBRARY_PATH=/home/runner/.apt/usr/lib/x86_64-linux-gnu:/home/runne
r/.apt/usr/lib/i386-linux-gnu:/home/runner/.apt/usr/lib:
INCLUDE_PATH=/home/runner/.apt/usr/include:/.apt/usr/include/x
-linux-gnu:
```

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

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

int main(void) {

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

putenv("HOME=/");

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

putenv("HOME");

printf("New home directory is %s\n", getenv("HOME"));
return 0;
}
```

```
> ./main
Home directory is /home/runner
New home directory is /
New home directory is (null)
```

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 euid: 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 maxim
  - 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

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

#define CMDLEN 80

int main(void) {
  char cmdstr[CMDLEN];
  printf("Enter command to run: ");
  fflush(stdout);
  fgets(cmdstr, CMDLEN, stdin);
  system(cmdstr);
  return 0;
}
```

```
> ./main
Enter command to run: ls -lia
total 20
    256 drwxr-xr-x 1 runner runner   26 Apr 13 01:20 .
9246086 drwxr-xr-x 1 runner runner 4096 Apr 13 01:05 ..
    257 drwxr-xr-x 1 runner runner   28 Apr  9 00:11 dir
    332 -rwxr-xr-x 1 runner runner 8544 Apr 13 01:20 main
    333 -rw-r--r-- 1 runner runner  226 Apr 13 01:20 main.c
```

More Examples #1: exit (1)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.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)

```c
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);
  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 */
```

More Examples #1: exit (3)

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

```
> ./main
normal termination, exit status = 7
abnormal termination, signal number = 6 (core file generate)
abnormal termination, signal number = 8 (core file generate)
```

More Examples #2: Orphan process (1)

```c
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <stdio.h>
#include <stdlib.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);

/* 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. */
```

More Examples #2: Orphan process (2)

```
    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);
}
```

```
hasoo$ gcc 05p2-mex2.c
hasoo$ ./a.out
hasoo$ second child, parent pid = 1
```

More Examples #3: fork

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


static void charatatime (char *str) {

  char *ptr;
  int c;

  /* set unbuffed */
  setbuf(stdout, NULL);

  for (ptr = str;c = *ptr++;)
    putc(c, stdout);

}
```

```c
int main(void) {
  pid_t pid;
  if((pid = fork()) < 0) {
    perror("fork error\n");
    exit(1);
  } else if (pid == 0)
    charatatime("output from child\n");
  else
    charatatime("output from parent\n");
  exit(0);
}
```

```
> ./main
output fromo uptapruetn tf
rom child
```

More Examples #4: fork

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

static void charatatime (char *str) {

  char *ptr;
  int c;

  /* set unbuffed */
  setbuf(stdout, NULL);

  for (ptr = str;c = *ptr++;)
    putc(c, stdout);

}
```

```c
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);
}
```

```
> ./main
output from parent
```

More Examples #5: exec (1)

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

char *env[] = {"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 ("~/bin/echoall", "echoall", "arg1", "arg2", (char *) 0, env) < 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, inhert enviroment */
    if (execlp("echoall", "echoall", "only 1 arg", (char *) 0) < 0) {
      fprintf(stderr, "execlp error\n");
      exit(1);
    }
  }
  exit(0);
}
```

More Examples #6: system (1)

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

/* without signal handling */
int mysystem(const char *cmd) {
  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)

```
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)

```c
int main(void) {
  int status;
  if ((status = mysystem("data")) < 0) {
    perror("mysystem() error\n");
    exit(1);
  }
  pr_exit(status);
  if ((status = mysystem("no such command")) < 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);
}
```

```
> ./main
sh: 1: data: not found
normal termination, exit status = 127
sh: 1: no: not found
normal termination, exit status = 127
normal termination, exit status = 44
```

More Examples #7: fork (1)

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

```c
/* 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 */
}
```

잘못 설명된 부분

- execl이 정상적으로 실행되면, exit(7)은 실행되지 않음
- execl이 실행되지 않은 경우에만, exit(7)이 실행됨
- exit()으로 반환한 숫자에 의해 잘못된 부분 체크 가능

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)

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <sys/times.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 #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);
}
```

More Examples #8: system (3)

```
static void do_cmd(char *cmd) { /* execute and time the "cmd" */
  struct tms tmsstart, tmsend;
  clock_t start, end;
  int status;
  fprintf(stderr, "\ncommand: %s\n", cmd);
  if ((start = times(&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)

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

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

  return 0;
}
```