PROCESS INFORMATION

UNIX Programming 2014 Fall by Euiseong Seo

Environment

- Each process has an environment, which is inherited from parent process
- Environment is a NULL-terminated array of strings
  - `extern char **environ;`
- Environment strings are of the form 'VAR=value'
  - Variable names are capitalized by convention

Reading Environment

- getenv(3) retrieves value associated with a variable

```
#include <stdlib.h>

char *getenv(const char *name);

char *value;

value=getenv("HOME");
if (value == NULL)
    printf("HOME not defined.\n");
else if (*value == '\0')
    printf("HOME defined but has no value.\n");
else
    printf("HOME=%s\n", value);
```

Adding Environment

- putenv(3) adds a var-value pair to environment

```
int putenv(const char *string)
```

```
putenv("HOME=/tmp");
```

- setenv(3) also adds a var-value pair to environment

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

```
int setenv(const char *name, const char *value, int overwrite)
```

Removing environment

- unsetenv(3) deletes a given variable from environment

```
#include <stdlib.h>

int unsetenv(const char *name);
```

- clearenv(3) clears all environment contents

Processing Arguments

- getopt(3) function provides a way to handle arguments
- Prototype

```
#include <unistd.h>

int getopt(int argc, char * const argv[], const char *optstring);

extern char *optarg;
extern int optind, opterr, optopt;
```

  - optind: index of the next argv element to be processed
  - optarg: pointer of argument for option

Processing Arguments

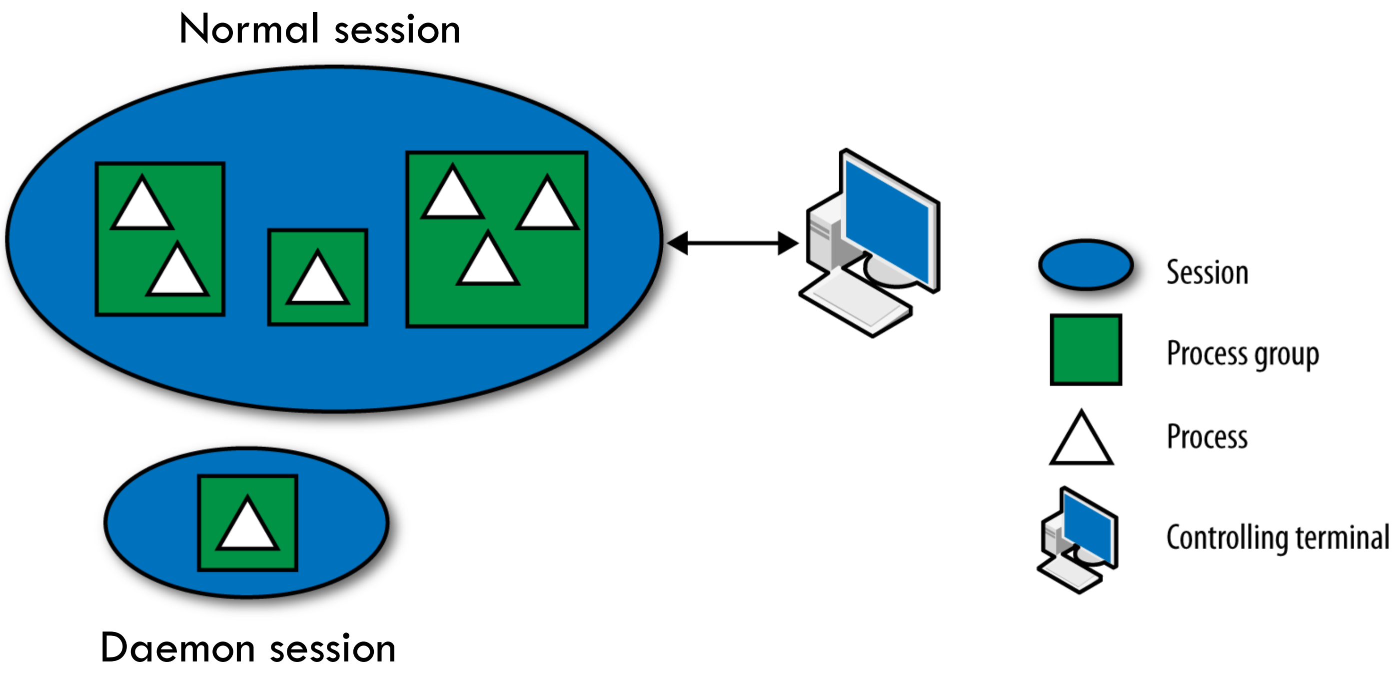
- Example

```
while((opt = getopt(argc, argv, "hvf:")) != -1)
{
    switch(opt)
    {
        case 'h':
            help();
            break;
        case 'v':
            version();
            break;
        case 'f':
            memcpy(file_name, optarg, 16);
            break;
    }
}
```

Process IDs and Process Group IDs

- Session ID
  - A process has a session ID
  - A session ID is set following the PID of session leader
  - Session leader = login shell
  - When a user log out every process in the session gets SIGQUIT signal
- Process group
  - A process belongs to a process group
  - A process group has a group leader
  - PGID = PID of group leader
  - Signals can be propagated to all processes in a group
  - This is for job controlling
  - All processes in this command belong to the same process group
    - cat ship-inventory.txt |grep booty |sort

Process IDs and Process Group IDs



Process IDs and Process Group IDs

- getpid(2) returns PID
- getppid(2) returns PID of parent process
- Prototype

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

pid_t getpid(void);
pid_t getppid(void);
```

- Example

```
#include <sys/types.h>
main()
{
    printf("My PID is %d.\n", getpid());
    printf("My PPID is %d.\n", getppid());
}
```

Process IDs and Process Group IDs

- getpgrp(2) returns process group ID
- setpgid(2) creates a new process group
- Prototype

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

Real and Effective IDs

- Real UID and GID can be obtained by getuid(2) and getgid(2), respectively
- Effective UID and GID can be obtained by geteuid(2) and getegid(2), respectively
- Prototype

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

Resource Limits

- UNIX enforces resource usage limit on each process
- Many resource limits are shown by ulimit(1)

```
euiseong@accept:~$ ulimit -a
core file size          (blocks, -c) 0
data seg size           (kbytes, -d) unlimited
scheduling priority             (-e) 0
file size               (blocks, -f) unlimited
pending signals                 (-i) 63531
max locked memory       (kbytes, -l) 64
max memory size         (kbytes, -m) unlimited
open files                      (-n) 1024
pipe size            (512 bytes, -p) 8
POSIX message queues     (bytes, -q) 819200
real-time priority              (-r) 0
stack size              (kbytes, -s) 8192
cpu time               (seconds, -t) unlimited
max user processes              (-u) 63531
virtual memory          (kbytes, -v) unlimited
file locks                      (-x) unlimited
euiseong@accept:~$
```

Resource Limits

- Hard limit
  - Root can lower or raise
  - Users can lower but not raise again
- Soft limit
  - User can lower or raise (up to hard limit)
  - Root can lower or raise
- Limits are inherited to the child processes

Resource Limits

Resource Macro	Meaning	Signal	Errno
RLIMIT_CORE	Maximum size of a core file in bytes that may be created by a process		
RLIMIT_CPU	Maximum amount of CPU time in seconds used by a process	SIGXCPU	
RLIMIT_DATA	Maximum size of process's heap in bytes		ENOMEM
RLIMIT_NOFILE	Maximum number of open file descriptors		
RLIMIT_STACK	Maximum size of a process's stack in bytes	SIGSEGV	
RLIMIT_NPROC	Maximum number of processes that can be created for a UID		EAGAIN

Resource Limits

- Prototype

```
#include <sys/time.h>
#include <sys/resource.h>

int getrlimit(int resource, struct rlimit *rlim);
int setrlimit(int resource, const struct rlimit *rlim);

struct rlimit {
    rlim_t rlim_cur;  /* Soft limit */
    rlim_t rlim_max;  /* Hard limit (ceiling for rlim_cur) */
};
```

Resource Limits

- Example

```
#include <sys/resource.h>
#include <unistd.h>
main()
{
    struct rlimit myrlim;

    getrlimit(RLIMIT_NOFILE, &myrlim);
    printf("I can only open %d files\n", myrlim.rlim_cur);
    myrlim.rlim_cur = 256;
    if(setrlimit(RLIMIT_NOFILE, &myrlim) == -1)
        perror("setrlimit");
    getrlimit(RLIMIT_NOFILE, &myrlim);
    printf("I can now open %d files.\n", myrlim.rlim_cur);
    printf("sysconf() says %d files.\n", sysconf(_SC_OPEN_MAX));
}
```

Time Usage

- You can determine the time usage of a process with times(2)
  - Time reported by times(2) is in clock ticks
  - You have to convert clock ticks to second
- Prototype

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

clock_t times(struct tms *buf);

struct tms {
    clock_t tms_utime;  /* user time */
    clock_t tms_stime;  /* system time */
    clock_t tms_cutime; /* user time of children */
    clock_t tms_cstime; /* system time of children */
};
```

Time Usage

- Types of time
  - Wall-clock time: time spent in real world
    - Return value of `times` shows elapsed wall-clock time from an arbitrary time point
  - User time: time spent in user-level
  - System time: time spent in kernel-level
  - User time of children: time spent by terminated and cleaned up children in user-level
  - System time of children: time spent by terminated and cleaned up children in kernel-level

Time Usage

- Example

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

main()
{
    int m;
    time_t t;
    struct tms mytms;
    clock_t time1, time2;
    double tick = sysconf(_SC_CLK_TCK);
    if((time1 = times(&mytms)) == -1)
        { perror("times"); exit(1); }
    for( m = 0 ; m < 99999; m++)
         { time(&t); }
    if((time2 = times(&mytms)) == -1)
        { perror("times"); exit(1); }
    printf("Real time: %.1f sec.\n", (time2-time1)/tick);
    printf("User time: %.1f sec.\n", mytms.tms_utime/tick);
    printf("Sys time: %.1f sec.\n", mytms.tms_stime/tick);
}
```

Current Directory

- getcwd(3) retrieves current working directory
- chdir(2) changes current working directory
- Prototype

```
#include <unistd.h>

char *getcwd(char *buf, size_t size);
int chdir(const char *path);
```

Current Directory

- Example

```
#include <sys/param.h>
#include <unistd.h>
main()
{
    char *dir;
    long pathmaxlen = pathconf("/", _PC_PATH_MAX);
    dir=getcwd((char *)NULL, pathmaxlen+1);
    if(dir==NULL)
       {perror("getcwd"); exit(1)}
    printf("CWD: %s\n", dir);
    free(dir);
    if(chdir("/tmp") == -1)
        perror("chdir");
    dir=getcwd((char *)NULL, pathmaxlen+1);
    if(dir==NULL)
        perror("getcwd");
    printf("CWD: %s\n", dir);
}
```

Shell Lab

- Skeleton of a shell is very simple

```
while(1) {
    print prompt
    read command
    process command
}
```

- Let's make a shell!