29/10/2023 – Fri

System Programming Notes

Chapter #3: Processes in Unix

→ A process is basic active entity in most OS's model.

3.1 Process Identification:

→ Each process has a process Id, parent process Id, user Id and group Id which are use for its identifications.

→ 'pid_t' is an unsigned integer type that represent a process id.

→ Synopsis:

```
#include <unistd.h>
pid_t getpid(void);
pid_t getppid(void);
```

- These functions return process id and the parent process id respectively.
- They cann't return an error.
- There is no guarantee that a 'pid_t' will fit into an 'int'

(2)

→ To each user System administrators provide a user Id and an integral group Id.

→ The most privileged user, super-user, system administrator, or root has a user Id of 0.

- $ sudo addgroup <group-Name>
- $ sudo usermod -aG

→ Synopsis:

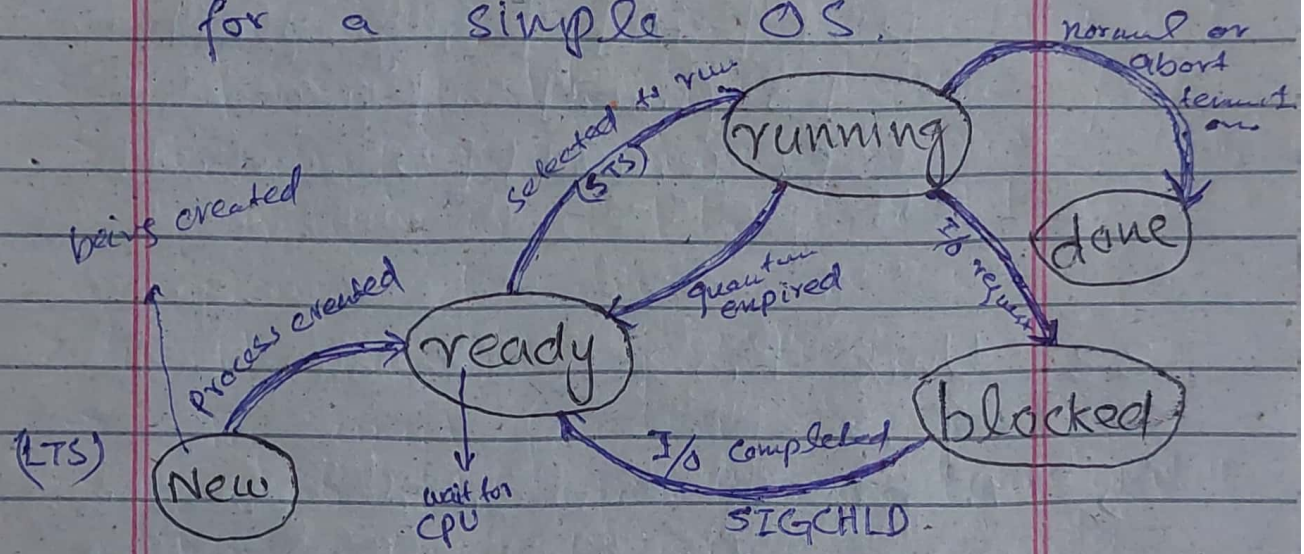
```
#include <unistd.h>
    gid_t  getegid (void);
    uid_t  geteuid (void);
    gid_t  getgid (void);
    uid_t  getuid (void);
```

- gid_t and uid_t are integral types representing group and user Ids.
- The 'getgid' and 'getuid' returns real Ids, and 'getegid' and 'geteuid' returns the effective Ids.
- None of these function returns error.

3.2 Process State:

→ The state of a process indicates its status at an time.

→ Below is state diagram for a simple OS.



→ A process perform I/O by requesting the services throug system call. During the execution of system call, the OS regain control of processor and can move processes to blocked state until operation completes.

→ A content switch is moving a process from running to and replacing it with another.

→ The 'ps' utility display the information about processes.

Synopsis:

PS [-aA] [-G grouplist]
[-o format] ... [-p proclist]
[-t termlist] [-U userlist].

3.3 Unix Process Creation and fork:

→ A process can create another process by calling fork. The caller becomes the parent and the created is called child process.

→ The fork function copies the parent memory image.

→ Synopsis

```
#include <unistd.h>
pid_t fork (void);
```

- returns 0 to child and child's pid to parent on success.

→ • On failure it returns -1 and set errno. (EAGAIN).

→ The output from the processes (child + parent) can appear in either order, and may not always be the same for running again and again in order.

3.4 The 'wait' function:

→ The wait function causes the caller to suspend execution untill a child's status becomes available or untill the caller receives a signal.

→ A process status become available after termination or after a process has been stoped.

→ Synopsis: #includ <sys/wait.h>.

- pid_t wait (int *stat_loc);
- pid_t waitpid (pid_t pid, int* stat_loc, int options);

- If 'pid' is -1, it will

wait for any child.
- pid is pointer to a location for returning a status and a flag specifying options.
- If pid > 0, it will wait for a child whose process id is pid.
- If pid < -1, it will wait for any child in process group specified by absolute value of pid.
- The 'options' parameter of waitpid is bitwise inclusive OR of one or more flags.
- On success these function returns pid of child.
- On failure it return -1 and set errno.

→ If a parent is not waiting for a child it is known as ~~orphan~~ zombie

→ If a parent terminates without waiting for its child, the child becomes an orphan.

→ 'stat_loc' argument of 'wait' or 'waitpid' is a pointer to integer variable, if it is not NULL, these functions store return status of child in this location.

→ Child return its status by exit, _exit, _Exit or return from main.

→ Parent only access the 8 least significant bits of child's return status.

→ POSIX specifies 6 Macros for testing child's return status. Each takes the status value returned by a child to 'wait' or 'waitpid' as a parameter.

→ Synopsis

```
#include <sys/wait.h>
i)   WIFEXITED(int stat_val);
ii)  WEXITSTATUS( " );
iii) WIFSIGNALED( " );
iv)  WTERMSIG( " );
v)   WIFSTOPPED( " );
vi)  WSTOPSIG( " );
```

- These are designed to be used in pairs.
- (i) returns non-zero true if process terminates normally.
- (i) == true Then: (ii) evaluates to the lower 8-bits return by child throug exit, _exit, _Exit or return.
- (iii) evaluates to non-zero true when child terminates because of an uncaught signal/ abnormal termination of child.
- (iii) == true Then: (iv) reason of abnormal termination, evaluates to number of signal that caused termination.
- (v) evaluates to non-zero true value if child is currently stopped.
- (v) == true Then: (vi) evaluates to number of signal that caused the child to stop/ reason of suspension/stopping the child.

3.5 The exec Function:

→ Fork creates image of callers. the exec family functions creates a new image other than that of caller.

→ Synopsis

```
#include <unistd.h>
extern char **environ;
int execl (const char* path,
    const char* filename, const
    char* arg[0], ..... , NULL);
int execlp(const char* filename,
    const char* arg[0], ... , NULL);
int execv (const char* path,
    char *const argv[]);
int execvp (const char* filename,
    char *const argv[]);
int execle(const char* path,
    const char* arg[0], ... , NULL);
int execve (const char* path,
    char *const* argv[], char *const
    envp[] );
```

- They dont return anything in success case

- These functions return -1 and set 'errno' if unsuccessfull.

→ The 'exec' with 'l' family is used if we know the number of CLA's at compile time.

→ The exec with 'v' family are used if we dont know the number of CLA at compile time.