Operating system Lab Exam

Assignment 1

- Fork is defined under unistd.h. it returns -1 on error.
- getpid() and getppid() are defined under <sys/types.h>
- wait(&status) and waitpid(processid,&status,options) are defined under <sys/wait.h>
- Normally the pparent process is executed before the child process.
- The child has its own unique process ID, and this PID does not match the ID of any existing process group.

* The child's parent process ID is the same as the parent's process ID.

* The child's set of pending signals is initially empty.

* Process resource utilizations and CPU time counters are reset to zero in the child.

* The child does not inherit its parent's memory locks.

* The child does not inherit semaphore adjustments from its parent.

* The child does not inherit record locks from its parent.

* The child does not inherit timers from its parent.

* The child does not inherit outstanding asynchronous I/O operations from its parent , nor does it inherit any asynchro□nous I/O contexts from its parent.

WAIT System call

Options in waitpid()

- >0: child process whose process id is equal to pid
- =0: child process whose group id is equal to calling process
- <-1: child process whose process group id is equal to absolute value of pid.
- =-1: wait for any child process.
- Wait is used for state changes in a child of the calling process, and obtain information about the child whose state has changed. A state change is considered to be: the child terminated; the child was stopped by a signal; or the child was resumed by a signal.In the case of a terminated child, performing a wait allows the system to release the resources associated with the child; if a wait is not performed, then the terminated child remains in a "zombie" state.
- Wait() returns the process id of the terminated child or returns -1 on error.
- Waitpid() returns the process id whose state has been changed else 0 is returned if WNOHANG is specified by existing pid but whose state has not been changed else -1 is returned on error.
- A child that terminates but has not been waited for becomes ZOMBIE.

Value of Options is or of zero or more things.

WNOHANG is returned immediately if no child has exited.

WUNTRACED is returned if an untraced child has stopped.Status for traced children which have stopped is provided even if this option is not specified

WCONTINUED previously stopped children that have been resumed by delivery of SIGCONT.

Macros which can be compared with status in wait() or waitpid():

- WIFEXITED – returns true if a child is terminated normally.
- WEXITSTAUS-returns the exit status of the child.Can be used only after WIFEXITED is retuurned.
- WIFSIGNALED- returns true if a child process was terminated by a signal.
- WTERMSIG-returns the signal number that caused the child process to terminate. Can be used only after WIFSIGNALED is returned.
- WIFSTOPPED-returns true if child was stopped by delivery of signal.it is possible only if the call was done by WUNTRACED.
- WSTOPSIG – returns the number of signal which caused child to stop.
- WIFCONTINUED -returns true if child process was resumed by SIGCONT.
- WCOREDUMP- returns true if child produced a core dump.Can be used after WIFSIGNALED is true.

Sleep delay means to pause forspecified number of seconds.It can be a floating number.

- The rand() function returns a pseudo-random integer in the range 0 to RAND_MAX inclusive (i.e., the mathematical range [0, RAND_MAX]).
- The srand() function sets its argument as the seed for a new sequence of pseudo-random integers to be returned by rand(). These sequences are repeatable by calling srand() with the same seed value. It does not return any value.
- If no seed value is provided, the rand() function is automatically seeded with a value of 1.
- Both srand and rand are defined under <stdlib.h>

Time runs the program COMMAND . When COMMAND finishes, time displays information about resources used by COMMAND (on the standard error output, by default). If COMMAND exits with non-zero status, time displays a warning message and the exit status.it is defined under time.h

exec() forms which take envp argument specify the environment for new process image while other forms of exec that do not take envp argument has an extern variable environ in calling process.

Argument +Environment lists =ARG_MAX

Both argc,argv and environ arrays are terminated by null pointer. This null termination pointer is not counted in argc. Files which are opened during calling process remain opened and none of their attributes are changed.

All signals which are either in defult state or those which are set to be caught in the calling process are set to default action in new process image. Signals which are to be ignored by calling process are said to be ignored by new process.
New process inherits process ID,parent process ID,group ID,current working directory,nice value,pending signals, semaphore adjustment values and current directory.

Exec does not return any value as new image is loaded over existing image.On failure it returns -1 and sets errno to one of the vaues:

E2BIG: number of bytes used by new process image 's argument and environment list is greater than ARG_MAX bytes

EACESS: new process denied execution or search permission denied for the directory.

ENAMETOOLING: length of path or file exceeds PATH_MAX

ENOENT: A component of path or file does not name an existing file or path.

ENOEXEC: the new process image has appropriate access permissions but it does not have proper format.

ENOMEM: the new process requires more memory than that is available.

ENOTDIR : A componet of new process 's image file path prefix is not a directory.

execve() and execle() are Async-signal safe while all othersare multithread safe if their environment variables are not modified.

All executable function ared defined in unistd.h file.

Signal Name Description -All of them are plced in signal.h

SIGABORT	*Process abort
SIGALRM	Alarm clock
SIGFPE	*Floating-point exception
SIGHUP	Hangup
SIGILL	*Illegal instruction
SIGINT	Terminal interrupt
SIGKILL	Kill (can’t be caught or ignored)
SIGPIPE	Write on a pipe with no reader
SIGQUIT	Terminal quit
SIGSEGV	Invalid memory segment access
SIGTERM	Termination
SIGUSR1	User-defined signal 1
SIGUSR2	User-defined signal 2
SIGCHLD	Child process has stopped or exited.
SIGCONT	Continue executing, if stopped.
SIGSTOP	Stop executing. (Can’t be caught or ignored.)
SIGTSTP	Terminal stop signal.
SIGTTIN	Background process trying to read.
SIGTTOU	Background process trying to write.

If a process receives one of these signals without first arranging to catch it, the process will be termi-
nated immediately. Usually, a core dump file is created. This file, called core and placed in the current directory, is an image of the process that can be useful in debugging.

killall , which allows you to send a signal to all processes running a specified command.This is useful when you do not know the PID, or when you want to send a signal to several different processes executing the same command.

signal is a function that takes two parameters, sig and function pointer .
The signal to be caught or ignored is given as argument sig . The function to be called when the specified signal is received is given as func . This function must be one that takes a single int argument and is of type void .

The signal function returns the previous value of the signal handler for the specified signal if there is one, or SIG_ERR otherwise, in which case, errno will be set to a positive value. errno will be set to EINVAL if an invalid signal is specified or an attempt is made to handle a signal that may not be caught or ignored, such as SIGKILL .

Sending signal may fail if target process is owned by another user.

The kill function sends the specified signal, sig , to the process whose identifier is given by pid . It returns 0 on success.kill will fail, return -1 , and set errno if the signal given is not a valid one (errno set to EINVAL), if it doesn't have permission (EPERM), or if the specified process doesn't exist (ESRCH).

struct msqid ds // for message queues
struct shmid ds // for shared segments
struct semid ds // for semaphores
They are defined in:-
#include <sys/ipc.h>
#include <sys/types.h>

converts a pathname and a project identifier to a (System V) IPC-key.
key_t ftok(const char *pathname, int proj id)

#include <sys/msg.h>-Header file needed for the following construct:-
struct message {
long type ;
char messagetext [MESSAGESI ZE];
};

int msgget(key t key, int msgflg)

- returns (creates) a message queue identifier associated with the value of the key argument.
- A new message queue is created if key has the value IPC PRIVATE.
- returns (creates) a message queue identifier associated with the value of the key argument to create a queue.
- the least significant bits of msgflg define the permissions of the message queue.

int msgsnd(int msqid, const void *msgp, size t msgsz, int msgflg)

struct msgbuf {
long mtype ; // m e s s a g e type - must be > 0
char mtext [MSGSZ]; // m e s s a g e data
};
ssize t msgrcv(int msqid, void *msgp, size t msgsz, long msgtyp,
int msgflg);

❿ receive a message msgp from a message queue with id msqid
❿ msgtyp is a strictly positive integer value.
 if msgtyp is zero, the first message is retrieved regardless its
type.
 This value can be used by the receiving process for message
selection.
 mesgsz specifies the size of the field mtext.
 By and large, msgflg is set to 0.
❿ msgtyp specifies the type of message requested as follows:
❿ if msgtyp=0 then the first message in the queue is read.
❿ if msgtyp > 0 then the first message in the queue of type msgtyp is read.
❿ if msgtyp < 0 then the first message in the queue with the lowest type value is read.

int msgctl(int msqid, int cmd, struct msqid ds *buf)
performs the control operation specified by cmd on the message queue with identifier msqid.

For Client-Server
#define MSGSIZE 256
#define PERMS 0666

PERMS-permission
IPC_CREAT-create a queue.

Semget() system call
```
   #include <sys/types.h>
     #include <sys/ipc.h>
     #include <sys/sem.h>
```

shmget()
int shmget(key t key, size t size, int shmflg)
#include <sys/ipc.h>
#include <sys/shm.h>

void *shmat(int shmid, const void *shmaddr, int shmflg)
 attaches the shared memory segment identified by shmid to
the address space of the calling process.
 If shmaddr is NULL, the OS chooses a suitable (unused)
address at which to attach the segment (frequent choice).
 Otherwise, shmaddr must be a page-aligned address at which
the attach occurs.

int shmdt(const void *shmaddr)
 detaches the shared memory segment located at the address
specified by shmaddr from the address space of the calling
process.

int shmctl(int shmid, int cmd, struct shmid ds *buf)
 performs the control operation specified by cmd on the shared
memory segment whose identifier is given in shmid.