Internals of Operating System (Practical No 3)

Prepared by: Khushboo Patel

Definition A

Write a C program that will print parent process id and child process id.

Mention error checking if child process is not created.

- Fork () - creates a child process.
- Available under <unistd.h>
- Duplicates the parent process
- Returns process id of child process (in integer)
- Does not take any argument.
- Ex, pid=fork()
- Return values:
- If **fork()** returns a negative value, the creation of a child process was unsuccessful.
- **fork()** returns a zero to the newly created child process.
- **fork()** returns a positive value, the ***process ID*** of the child process, to the parent. The returned process ID is of type **pid_t** defined in **sys/types.h**. Normally, the process ID is an integer.

In these cases fork() may fail:

Fork() will fail and no child process will be created if:

1. The system-imposed limit on the total number of processes under execution would be exceeded. This limit is configuration-dependent.
2. The system-imposed limit MAXUPRC (<sys/param.h>) on the total number of processes under execution by a single user would be exceeded.
3. There is insufficient swap space for the new process.

Get child/parent process id

Child process id : int getpid()

Parent process id : int getppid()

printf(“calling process id is process id:%d”,getpid());

Example

```
int main(){
  pid_t pid, ppid;

  //get the process'es pid
  pid = getpid();

  //get the parent of this process' pid
  ppid = getppid();



  printf("My pid is: %d\n",pid);
  printf("My parent's pid is %d\n", ppid);

  return 0;
}
```

Execute program 1 and discuss output

```
#include  <stdio.h>
#include  <string.h>
#include  <sys/types.h>
#define   MAX_COUNT  200
#define   BUF_SIZE   100
void  main(void){
     pid_t  pid;
     int    i;
     char   buf[BUF_SIZE];
     fork();
     pid = getpid();
     for (i = 1; i <= MAX_COUNT; i++) {
          sprintf(buf, "This line is from pid %d, value = %d\n", pid, i);
          write(1, buf, strlen(buf));
     } }
```

Attempt following questions and discuss some points

1. Which process complete first?
2. What is role of CPU scheduler?
3. Discuss about value of ‘id’ in both the processes.

Definition B

In continuation of part (a), write a C program where parent process wait for child process to terminate.

Solution: wait()

Wait()

- The **wait**() system call suspends execution of the calling process until **one of its children** terminates.
- All of these system calls are used to wait 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
- Differentiate: waitpid() and waitid()

Example

```
#include <sys/wait.h>
int main(){
  pid_t c_pid, pid;
  int status;
  c_pid = fork();
  if( c_pid == 0 ){
    //child
    pid = getpid();
    printf("Child: %d: I'm the child\n", pid, c_pid);
    printf("Child: sleeping for 2-seconds, then exiting with status 12\n");
    sleep(2);
}
```

```
else if (c_pid > 0){
    //parent

    //waiting for child to terminate
    pid = wait(&status);

    if ( WIFEXITED(status) ){
      printf("Parent: Child exited with status: %d\n", WEXITSTATUS(status));
    }

  }else{
    //error: The return of fork() is negative
    perror("fork failed");
    //exit(2);
  }

  return 0; //success
}
```

Exercise

1. Perform fork() for 3 times and wait() parent process. Observe behaviour of child processes.
2. In continuation of (1), sleep child process for 10 seconds and observe behaviour.

Definition C

Write a C program using execvp() system call which will count the characters from file ‘wc.txt’, using program ‘p.c’.

execvp()

execvp : Using this command, the created child process does not have to run the same program as the parent process does. The **exec** type system calls allow a process to run any program files, which include a binary executable or a shell script . **Syntax:**

```
int execvp (const char *file, char *const argv[]);
```

file: points to the file name associated with the file being executed.

argv: is a null terminated array of character pointers.

execvp() example

```
#include <stdio.h>  // perror()
#include <stdlib.h> // EXIT_SUCCESS, EXIT_FAILURE

int main(void) {
  char *const cmd[] = {"ls", "-l", NULL};
  execvp(cmd[0], cmd);
  perror("Return from execvp() not expected");
  exit(EXIT_FAILURE);
}
```