Operating Systems Lab

Lab # 06

Wait and Sleep System Call

Instructor: Engr. Muhammad Usman

Email: usman.rafiq@nu.edu.pk

Course Code: CL2006

Department of Computer Science,

National University of Computer and Emerging Sciences FAST Peshawar Campus

Contents

Process Waiting States

sleep() system call

The sleep() call can be used to cause delay in execution of a program. The delay can be provided as an integer number (representing number of seconds). Usage is simply sleep(int), which will delay the process execution for int seconds. To use sleep(), you will require the ***unistd.h*** C library.

Exercise

Write a C program that can display a count from 10 to 0 (reverse order) using a for or a while loop. Each number should be displayed after delay of 1 second.

sleepExercise.c

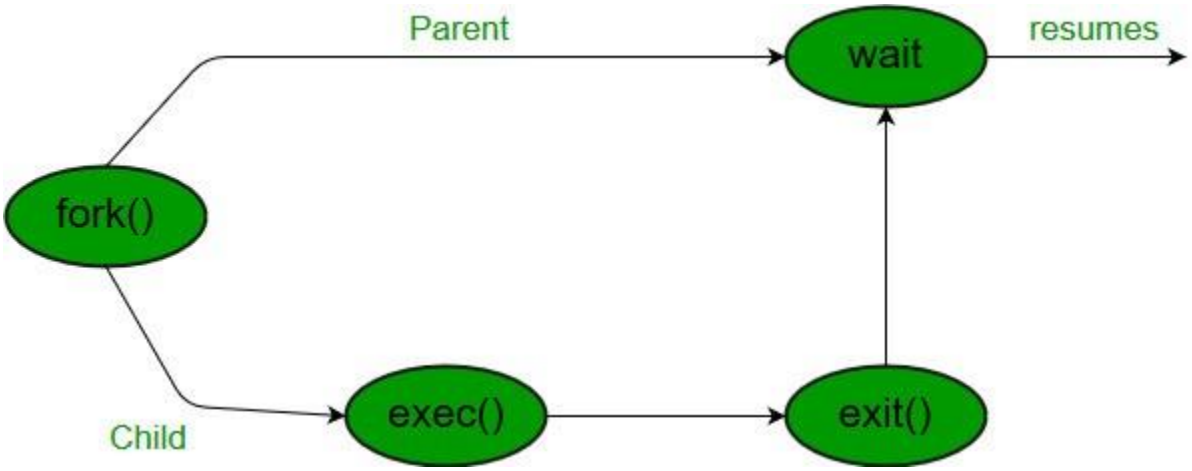
```
#include <unistd.h>
#include <stdio.h>
int main()
{
for(int i=10; i>=0; i--)
{
printf("%d\n",i);
sleep(2);
}
}
```

wait() system call

Suspend execution until a child process exits. Wait returns the exit status of that child.

A call to wait() blocks the calling process until one of its child processes exits or a signal is received. After child process terminates, parent ***continues*** its execution after wait system call instruction.
Child process may terminate due to any of these:

- It calls exit();
- It returns (an int) from main
- It receives a signal (from the OS or another process) whose default action is to terminate.



Syntax in c language:

```
#include
#include
// take one argument status and returns
// a process ID of dead children.
pid_t wait(int *stat_loc);
```

- If any process has more than one child processes, then after calling wait(), parent process has to be in wait state if no child terminates.
- If only one child process is terminated, then return a wait() returns process ID of the terminated child process.
- If more than one child processes are terminated than wait() reap any ***arbitrarily child*** and return a process ID of that child process.
- When wait() returns they also define **exit status** (which tells our, a process why terminated) via pointer, If status are not **NULL**.

- If any process has no child process then wait() returns immediately “-1”. NOTE: “This codes does not run in simple IDE because of environmental problem so use terminal for run the code” .

Examples:

wait1.c

```
// C program to demonstrate working of wait()
#include<stdio.h>
#include<stdlib.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
   pid_t cpid;
   if (fork()== 0)
exit(0);                    /* terminate child */
   else
cpid = wait(NULL);       /* reaping parent */
   printf("Parent pid = %d\n", getppid());
   printf("Child pid = %d\n", cpid);
   return 0;
}
```

Output:

```
Parent pid = 5150
Child pid = 6046
```

wait2.c

```
// C program to demonstrate working of wait()
#include<stdio.h>
#include<sys/wait.h>
#include<unistd.h>
int main()
{
if (fork()== 0)
printf("HC: hello from child\n");
else
{
printf("HP: hello from parent\n");
wait(NULL);
printf("CT: child has terminated\n");
}
        printf("Bye\n");
return 0;
}
```

Output: depend on environment

```
HC: hello from child
Bye
HP: hello from parent
CT: child has terminated
     (or)
HP: hello from parent
HC: hello from child
HC: Bye
CT: child has terminated    // this sentence does
Bye                         //not print before HC because of wait.
```

wait3.c

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include<sys/wait.h>
int main()
{
int pid;
pid = fork();
if(pid==0)
{
printf("Child is running...\n");
}
else
{
```

```
printf("Parent is running...\n");
wait(NULL);
printf("Child terminated\n");
}
printf("Done\n");
return 0;
}
```

Output: depend on environment

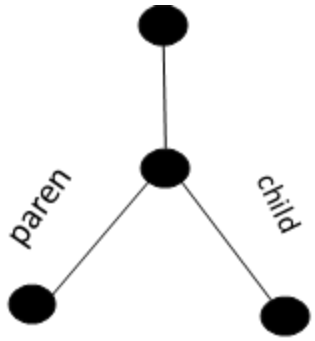
Parent is running...

Child is running...

Done

Child terminated

Done



Parent running

Child running
Done

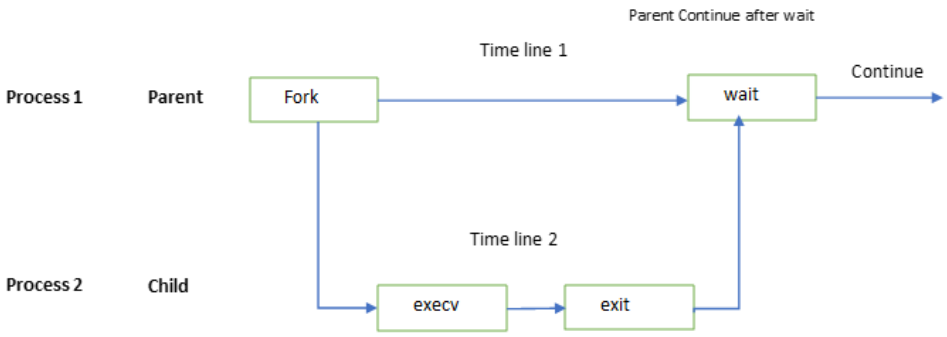
Child terminated
Done

Terminal Inputs
fork

parent: -----> wait sta
child ---->exec() --->e
parent: ---

Parent and Child Processes

Case # 01: If parent process executes first



Case # 02: If child process executes first