##

Experiment No. 3

##

Author: Vidyut Chakrabarti
Semester/section: IV Sem/ B
Roll no.: 68
Date of execution: 09/02/24

Source code: prac3_b468.c

##

Aim: To write C programs to implement the process control system calls.

Problem Statement:

Write a menu driven program in C that uses different process control commands to execute following tasks to –

[1] Create processes that execute the address space of the parent

[2] Create processes that do not execute the address space of the parent.

[3] Display IDs for parent and child processes.

[4] Execute processes swapping running process with another one when

- (a) absolute path is required,

- (b) absolute path is optional, include all variations

[5] Create zombie and orphan processes.

[6] Terminate the process – (a) with cleanup, (b) without cleanup

===

Menu Driven C program: prac3_b468.c

===

```
#include<sys/types.h>
#include<sys/stat.h>
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>

int main(){
   int s, pid;
```

```
printf("Enter 1 for creating processes that execute the address space of
the parent\nEnter 2 for executing process that does not execute the
address space of the parent process\nEnter 3 for displaying IDs of parent
and child processes.\nEnter 4 for executing execl.\nEnter 5 for executing
execlp\nEnter 6 for executing execle\nEnter 7 for executing execv\nEnter 8
for executing execvp\nEnter 9 for executing execvpe\nEnter 10 for creating
zombie process.\nEnter 11 for creating an orphan process.\nEnter 12 to
list plausible orphan processes.\n");

  printf("Enter your choice: ");
  scanf("%d", &s);
  switch(s){

/** [1] Create processes that execute the address space of the parent. **/

    case 1:
      printf("Executing Fork...\n");
      pid = fork();
      if(pid==0){
      printf("Child created successfully.\n");
      printf("pid of child: %d\n", getpid());
      printf("parent(pid): %d\n", getppid());
      printf("Hello, I am the child executing...\n");
      }
      else{
      printf("parent continues... \n");
      printf("grandparent(pid): %d\n", getppid());
      printf("pid of parent: %d\n", getpid());
      printf("Hello, this is the parent executing...\n");
      }
      printf("TERMINATING\n");
printf("==============================================================\n");
     break;

/** [2] Create processes that does not execute the address space of the parent.
**/
    case 2:
      printf("VFork execution...\n");
      pid = vfork();
      if(pid==0){
      printf("Child created successfully.\n");
      printf("pid of child: %d\n", getpid());
      printf("parent(pid): %d\n", getppid());
      printf("Child executing.\n");
      sleep(5);}
      else{
      printf("Parent continues... \n");
      printf("grandparent(pid): %d\n", getppid());
      printf("pid of parent: %d\n", getpid());
      printf("Parent executing.\n");}
      printf("TERMINATING\n");
printf("===============================================================\n");
      break;

/** [3] Display IDs for parent and child processes. **/

    case 3:
      printf("Executing Fork...\n");
      pid = fork();
      if(pid==0){
      printf("Child created successfully.\n");
      printf("pid of child: %d\n", getpid());
      printf("parent(pid): %d\n", getppid());
      printf("Hello, I am the child executing...\n");
      }
      else{
      printf("parent continues... \n");
      printf("grandparent(pid): %d\n", getppid());
      printf("pid of parent: %d\n", getpid());
      printf("Hello, this is the parent executing...\n");
      }
      printf("TERMINATING\n");
 printf("===============================================================\n")
      break;

/** [4] EXECL COMMAND EXECUTION (PATH REQUIRED) **/

Case 4:
    printf("EXECUTING EXECL:\n");
      const char *path = "/usr/bin/ls";
      const char *arg2 = "-l";
      const char *arg3 = "prac3_b468.c";
      printf("Finding prac3_b468.c in directory: \n");
      execl(path, path, arg2 , arg3 ,NULL);

  //arg1 : Path, every argument list must be terminated by NULL.
 // beacuse we have overwritten this address space with execl.
//therefore the next line doesn't execute....

      printf("\nTask over..........");
printf("===============================================================\n");
     break;

/** [5] EXECLP COMMAND EXECUTION (PATH NOT REQUIRED).  **/

    case 5:
      printf("EXECUTING EXECLP:\n");
// only filename no path, execlp will search for path automatically.

      const char *file = "ls";
      const char *a2 = "-l";
      const char *a3 = "prac3_b468.c";
      printf("Finding prac3_b468.c in directory: \n");
      execlp(file, file, a2 , a3 ,NULL);
      printf("\nTask over.........");
printf("==============================================================\n");
      break;

/** [6] EXECLE COMMAND EXECUTION (PATH REQUIRED).  **/
case 6:
  // which bash
  //echo $PATH

 printf("EXECLE EXECUTION:\n");

 // we need to exeute inside shell
 const char *pathname = "/usr/bin/bash";

 // initializing the enivronment variables
 const char *ag2 = "echo $ENV1 $ENV2!";

 const char *envp[] = {"ENV1=HELLO","ENV2=WORLD! THIS IS VIDYUT",NULL};
  // setting the env variables

  execle(pathname, pathname,"-c",ag2,NULL,envp);
   //-c flag needed for bash
  printf("\nTask over.........");

printf("==============================================================\n");
      break;

/** [7] EXECV COMMAND EXECUTION (PATH REQUIRED).  **/
Case 7:

      printf("EXECV EXECUTION: \n");

      const char *pn = "/usr/bin/ls";
      printf("Listing prac3_b468.c via execv: \n");
      char *const args[] = {"/usr/bin/ls","-l","prac3_b468.c",NULL};
//using array for execv

      execv(pn ,args);
      printf("\nTask over.........");

printf("==============================================================\n");
     break;

/** [8] EXECVP COMMAND EXECUTION (PATH NOT REQUIRED).  **/

printf("EXECVP EXECUTION:\n");
```

```c
      const char *filename = "ls";
      char *const arguments[] = {"ls","-l","prac3_b468.c",NULL};
      printf("Listing prac3_b468.c via execvp: \n");

      execvp(filename, arguments);
      printf("\nTask over..........");

printf("=============================================================\n");
      break;

/** [9] EXECVPE COMMAND EXECUTION (PATH NOT REQUIRED).  **/

case 9:

      printf("EXECVPE DEMO :\n");
      const char *fn = "/bin/bash";
    const char *ar[] = {"echo $ENVP1 $ENVP2",NULL};
    const char *envpe[] = {"ENVP1=HELLO","ENVP2=WORLD! I AM B4_68.",NULL};

      execvpe(fn,fn,"-c",ar,envpe);
      printf("\nTask over..........\n");

printf("=============================================================\n");
      break;

/** [10] CREATE A ZOMBIE PROCESS. **/
// Executes zombie68.c

case 10:

printf("Creating a zombie process in the background by executing
zombie.c.\n");

      system("gcc zombie68.c -o ./zombie.out");
      printf("Initiating zombie...\n");
      system("./zombie.out &");
printf("=============================================================\n");
      break;

/** [11] CREATE AN ORPHAN PROCESS. **/
// Executes orphanb4_68.c

case 11:
     printf("Executing an orphan process by running orphanb4_68.c...\n ");
     system("gcc orphanb4_68.c -o orphan.out");
     system("./orphan.out");
printf("==================================================================
\n");
      break;
```

```
/** [12] CHECKING FOR AN ORPHAN PROCESS **/
// runs orphancheck.sh

    case 12:
      printf("Checking for orphan processes..\n");
      system("./orphancheck.sh");
printf("================================================================\n
");
      break;
    default:
      printf("Invalid input.");
      printf("==================================================\n");
  }
  return 0;
 }

##########################################################################

/** zombie68.c **/
/** [10] CREATE A ZOMBIE PROCESS. **/
/** This program is executed from the main prac3_b468.c program to
initialize a zombie process in the background. **/

#include<stdio.h>
#include<unistd.h>
#include<sys/types.h>
#include <sys/wait.h>
#include <stdlib.h>

 int main (void){
 int pid;
 pid = fork();
 if(pid != 0){
 printf("Shh.. Main is sleeping. \n ");

 while(1){
 sleep(1000);
 }}

 else{
 printf("In child()...\n");
 exit(42);
 }
 printf("In main..\n");
 return 0;
}

##########################################################################
/** orphanb4_68.c **/
/** [11] CREATE AN ORPHAN PROCESS. **/
/** This program is executed from the main prac3_b468.c program to
initialize an orphan process. **/

#include<sys/types.h>
#include<sys/stat.h>
#include<sys/wait.h>
#include<stdio.h>
#include<unistd.h>
#include<string.h>
#include<stdlib.h>

int main(){
pid_t p;
int status;
p = fork();
if(p==0){
printf("CHILD PROCESS....\n");
exit(0);
}
else if(p>0){
printf("\t PARENT PROCESS...\n");
wait(&status);
printf("\t PARENT WAITING....\n");
if(WIFEXITED(status)){
printf("\n\t CHILD TERMINATED: %d\n", WEXITSTATUS(status));
}
printf("PARENT TERMINATES...\n");
}
else if(p == -1){
printf("\n FAILED: Unsuccessful fork()\n");
}
return 0;
}

###########################################################################

/** orphancheck.sh **/
/** [12] CHECKING FOR AN ORPHAN PROCESS. **/
/** This shell script is run when case 12 Is chosen form prac3_b468.c
program to look for an orphan process. **/

#!/usr/bin/sh
orphans=$(ps -ef | awk -v user=vidyut '$1 == user && $3 == 1 {print $2}')
echo "Plausible orphans are: $orphans"

############################################################################
```

EXECUTION TRACE:

```
============================================================================
```

[1] Create processes that execute the address space of the parent.
[3] Display IDs for parent and child processes.

```
vidyut@vidyut-VirtualBox:~/Desktop/B_68$ ./prac3.out
Enter 1 for creating processes that execute the address space of the
parent
Enter 2 for executing process that does not execute the address space of
the parent process
Enter 3 for displaying IDs of parent and child processes.
Enter 4 for executing execl.
Enter 5 for executing execlp
Enter 6 for executing execle
Enter 7 for executing execv
Enter 8 for executing execvp
Enter 9 for executing execvpe
Enter 10 for creating zombie process.
Enter 11 for creating an orphan process.
Enter 12 to list plausible orphan processes.
Enter your choice: 1
Executing Fork...
Child created successfully.
pid of child: 2340
parent continues...
parent(pid): 2339
Hello, I am the child executing...
TERMINATING
==============================================================
grandparent(pid): 2163
pid of parent: 2339
Hello, this is the parent executing...
TERMINATING
==============================================================
```

[2] Create processes that does not execute the address space of the parent.
[3] Display IDs for parent and child processes.

```
vidyut@vidyut-VirtualBox:~/Desktop/B_68$ ./prac3.out
Enter 1 for creating processes that execute the address space of the
parent
Enter 2 for executing process that does not execute the address space of
the parent process
Enter 3 for displaying IDs of parent and child processes.
…..
Enter 12 to list plausible orphan processes.
Enter your choice: 2
VFork execution...
Child created successfully.
pid of child: 2401
parent(pid): 2400
```

```
Child executing.
TERMINATING
==============================================================
Parent continues...
grandparent(pid): 2163
pid of parent: 2400
Parent executing.
TERMINATING
*** stack smashing detected ***: terminated
Aborted (core dumped)
==============================================================
```

[4] EXECL COMMAND EXECUTION (PATH REQUIRED)

```
Enter 1 for creating processes that execute the address space of the
parent
...
Enter 4 for executing execl
Enter 5 for executing execlp

...
Enter 12 to list plausible orphan processes.
Enter your choice: 4
EXECUTING EXECL:
Finding prac3_b468.c in directory:
-rw-rw-r-- 1 vidyut vidyut 3484 Feb 15 18:39 prac3_b468.c
=======================================================================
```

[5] EXECLP COMMAND EXECUTION (PATH NOT REQUIRED).

```
Enter 1 for creating processes that execute the address space of the
parent
...
Enter 5 for executing execlp
...
Enter 12 to list plausible orphan processes.
Enter your choice: 5
EXECUTING EXECLP:
Finding prac3_b468.c in directory:
-rw-rw-r-- 1 vidyut vidyut 3934 Feb 15 18:50 prac3_b468.c
=======================================================================
```

[6] EXECLE COMMAND EXECUTION (PATH REQUIRED).

```
Enter 1 for creating processes that execute the address space of the
parent
...
Enter 6 for executing execle
...
Enter 12 to list plausible orphan processes.
Enter your choice: 6
EXECLE EXECUTION:
HELLO WORLD! THIS IS VIDYUT!
======================================================================

[7] EXECV COMMAND EXECUTION (PATH REQUIRED).

Enter 1 for creating processes that execute the address space of the
parent
...
Enter 7 for executing execv
...
Enter 12 to list plausible orphan processes.
Enter your choice: 7
EXECV EXECUTION:
Listing prac3_b468.c via execv:
-rw-rw-r-- 1 vidyut vidyut 4888 Feb 15 19:13 prac3_b468.c

======================================================================

[8] EXECVP COMMAND EXECUTION (PATH NOT REQUIRED).

Enter 1 for creating processes that execute the address space of the
parent
...
Enter 8 for executing execvp
...
Enter 12 to list plausible orphan processes.
Enter your choice: 8
EXECVP EXECUTION:
Listing prac3_b468.c via execvp:
-rw-rw-r-- 1 vidyut vidyut 5258 Feb 15 19:22 prac3_b468.c
====================================================================

[9] EXECVPE COMMAND EXECUTION (PATH NOT REQUIRED).

Enter 1 for creating processes that execute the address space of the
parent
...
Enter 9 for executing execvpe
Enter 10 for creating zombie process.
Enter 11 for creating an orphan process.
Enter 12 to list plausible orphan processes.
Enter your choice: 9
EXECVPE DEMO :
HELLO WORLD! I AM B4_68.
=====================================================================

[10] CREATE A ZOMBIE PROCESS.
// Executes zombie68.c
```

```
Enter 1 for creating processes that execute the address space of the
parent
...
Enter 10 for creating zombie process.
Enter 11 for creating an orphan process.
Enter 12 to list plausible orphan processes.
Enter your choice: 10
Creating a zombie process in the background by executing zombie.c.
Initiating zombie...
Shh.. Main is sleeping.
In child()...
=====================================================================
```

[11] CREATE AN ORPHAN PROCESS.
// Executes orphanb4_68.c

```
Enter 1 for creating processes that execute the address space of the
parent
...
Enter 11 for creating an orphan process.
Enter 12 to list plausible orphan processes.
Enter your choice: 11
Executing an orphan process by running orphanb4_68.c...
       PARENT PROCESS...
CHILD PROCESS....
       PARENT WAITING....

       CHILD TERMINATED: 0
PARENT TERMINATES...
====================================================================
```

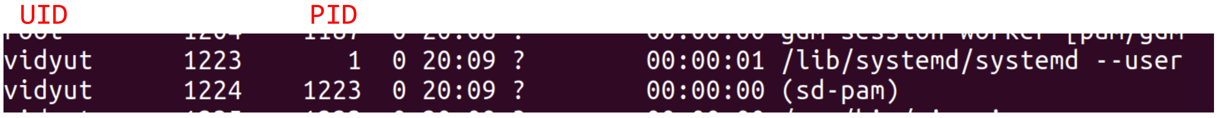
[12] CHECKING FOR AN ORPHAN PROCESS
// runs orphancheck.sh

```
Enter 1 for creating processes that execute the address space of the
parent
Enter 2 for executing process that does not execute the address space of
the parent process
Enter 3 for displaying IDs of parent and child processes.
Enter 4 for executing execl.
Enter 5 for executing execlp
Enter 6 for executing execle
Enter 7 for executing execv
Enter 8 for executing execvp
Enter 9 for executing execvpe
Enter 10 for creating zombie process.
Enter 11 for creating an orphan process.
Enter 12 to list plausible orphan processes.
Enter your choice: 12
```

```
Checking for orphan processes..
Plausible orphans are: 1223
1261
=================================================================

/** OUTPUT FROM ps -ef showing orphan process. **/

 UID                 PID
```



```
vidyut       1223       1  0 20:09 ?        00:00:01 /lib/systemd/systemd --user
vidyut       1224    1223  0 20:09 ?        00:00:00 (sd-pam)
```

```
==================================================================

[13] TERMINATING A PROCESS WITH CLEANUP (USING KILL)

Before kill
```

```
vidyut@vidyut-VirtualBox:~$ ps -u
USER      PID  %CPU  %MEM   VSZ      RSS TTY     STAT  START   TIME COMMAND
vidyut    1274   0.0    0.0    171044   6016 tty2    Ssl+  20:07  0:00 /usr/libexec/
vidyut    1288   0.0    0.2    231700  15616 tty2    Sl+   20:07  0:00 /usr/libexec/
vidyut    2217   0.0    0.0    20312    5632 pts/0   Ss    20:08  0:00 bash
vidyut    2703   0.0    0.0    20312    5760 pts/0   S+    20:45  0:00 bash
vidyut    2880   0.0    0.0    2776     1280 pts/0   S     21:02  0:00 ./zombie.out
vidyut    2881   0.0    0.0    0        0 pts/0      Z     21:02  0:00 [zombie.out]
vidyut    2902   0.2    0.0    19664    4992 pts/1   Ss    21:02  0:00 bash
vidyut    2913   0.0    0.0    21328    3456 pts/1   R+    21:03  0:00 ps -u
==========================================================================


After kill:
vidyut@vidyut-VirtualBox:~$ kill 2880
vidyut@vidyut-VirtualBox:~$ ps -u

USER      PID  %CPU  %MEM   VSZ      RSS TTY     STAT  START   TIME COMMAND
vidyut    1274   0.0    0.0    171044   6016 tty2    Ssl+  20:07  0:00 /usr/libexec/
vidyut    1288   0.0    0.2    231700  15616 tty2    Sl+   20:07  0:00 /usr/libexec/
vidyut    2217   0.0    0.0    20312    5632 pts/0   Ss    20:08  0:00 bash
vidyut    2703   0.0    0.0    20312    5760 pts/0   S+    20:45  0:00 bash
vidyut    2902   0.2    0.0    19664    4992 pts/1   Ss    21:02  0:00 bash
vidyut    2913   0.0    0.0    21328    3456 pts/1   R+    21:03  0:00 ps -u
===========================================================================
```

[14] TERMINATING A PROCESS WITHOUT CLEANUP (USING SEGKILL(9))

```
BEFORE KILL:
vidyut@vidyut-VirtualBox:~$ ps -u

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND

vidyut      1267  0.0  0.0 171044  5888 tty2     Ssl+ 20:09   0:00 /usr/libexec/
vidyut      1291  0.0  0.2 231700 15488 tty2     Sl+  20:09   0:00 /usr/libexec/
vidyut      2411  0.0  0.0  19792  4992 pts/1    Ss+  20:19   0:00 bash
vidyut      5164  0.0  0.0  19792  4992 pts/0    Ss   21:50   0:00 bash
vidyut      5269  0.0  0.0   2776  1280 pts/1    S    21:58   0:00 ./zombie.out
vidyut      5270  0.0  0.0      0     0 pts/1    Z    21:58   0:00 [zombie.out]
vidyut      5275  0.0  0.0  21328  3456 pts/0    R+   21:59   0:00 ps -u
==============================================================================

vidyut@vidyut-VirtualBox:~$ kill -9 5269
vidyut@vidyut-VirtualBox:~$ ps -u

USER         PID %CPU %MEM    VSZ   RSS TTY      STAT START   TIME COMMAND

vidyut      1267  0.0  0.0 171044  5888 tty2     Ssl+ 20:09   0:00
/usr/libexec/gdm-wayland-session env GNOME_SHELL_SESSION_MODE=ubuntu
/usr/bin/gnome
vidyut      1291  0.0  0.2 231700 15488 tty2     Sl+  20:09   0:00
/usr/libexec/gnome-session-binary --session=ubuntu

vidyut      2411  0.0  0.0  19792  4992 pts/1    Ss+  20:19   0:00 bash
vidyut      5164  0.0  0.0  19792  4992 pts/0    Ss   21:50   0:00 bash
vidyut      5276  0.0  0.0  21328  3456 pts/0    R+   21:59   0:00 ps -u

==========================================================================
```