Department of

Computer Science and Engineering

Tezpur University

Operating System Lab

Submitted by:
Sumsum Gogoi
CSM23036
MCA 2nd sem

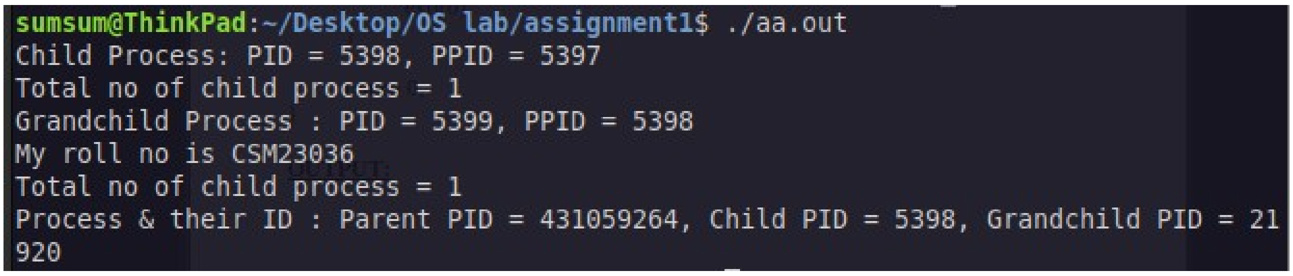
ASSIGNMENT 1-

1. A) Write a C program to create a child process using the system call fork( ). From the child process, display the PID and PPID and then call again the fork( ) to create a grandchild and engage him to display your roll no. From parent display the PID and PPID of all the processes and display the count of total no. of child processes created also the number of grandchild processes created. Demonstrate the use of exit(0) and exit(1).

CODE:

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

int main()
{

        pid_t parent_pid, child_pid, grandchild_pid;
        int child_count = 0, grandchild_count = 0;

        child_pid = fork();

        if (child_pid < 0)
        {
                perror("Error creating a child process");
                exit(1);
        }
        else if (child_pid == 0)
        {
                child_count++;
                child_pid = getpid();
                parent_pid = getppid();
                printf("Child Process: PID = %d, PPID = %d\n", child_pid, parent_pid);
                printf("Total no of child process = %d\n", child_count);

                grandchild_pid = fork();

                if (grandchild_pid < 0)
                {
                        perror("Error creating grandchild process");
                        exit(1);
                }
                else if (grandchild_pid == 0)
                {
                        grandchild_count++;
                        grandchild_pid = getpid();
                        child_pid = getppid();
                        printf("Grandchild Process : PID = %d, PPID = %d \n",
grandchild_pid, child_pid);
                        printf("My roll no is CSM23036\n");
                        printf("Total no of grand child process = %d\n",
grandchild_count);
                        exit(0);
                }
                else
                {
                        wait(NULL);
                        exit(0);
                }
        }
        else
        {
                wait(NULL);
                exit(0);
        }

        return 0;
}
```

<u>OUTPUT:</u>



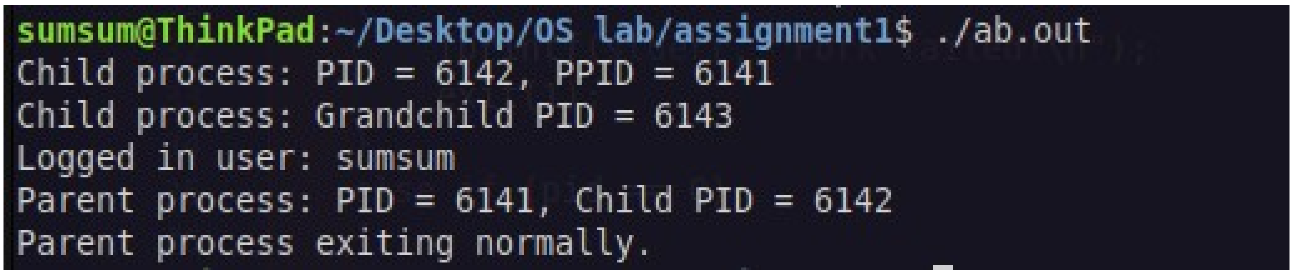
```
sumsum@ThinkPad:~/Desktop/OS lab/assignment1$ ./aa.out
Child Process: PID = 5398, PPID = 5397
Total no of child process = 1
Grandchild Process : PID = 5399, PPID = 5398
My roll no is CSM23036
Total no of child process = 1
Process & their ID : Parent PID = 431059264, Child PID = 5398, Grandchild PID = 21
920
```

1. B) Write a C program like the assignment 1(a). But here use the system call wait() system to synchronize the execution of parent program in your program until child process finishes. Now write a function to print to find out who is logging in the local machine that is partially equivalent to the cmd "w" or "who" and engage the grandchild to print the output into the stdout.

CODE:

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

void print_logged_in_users()
{
    struct utmp *utmp_entry;
    setutent();

    while ((utmp_entry = getutent()) != NULL)
    {
        if (utmp_entry->ut_type == USER_PROCESS)
        {
            printf("Logged in user: %s\n", utmp_entry->ut_user);
        }
    }

    endutent(); // Close utmp file
}

int main()
{
    pid_t pid, grandpid;

    // Create first child process
    pid = fork();

    if (pid < 0)
    {
        fprintf(stderr, "Fork failed!\n");
        exit(1);
    }
    else if (pid == 0)
    {
        // Child process
        printf("Child process: PID = %d, PPID = %d\n", getpid(), getppid());

        // Create grandchild process
        grandpid = fork();

        if (grandpid < 0)
        {
            fprintf(stderr, "Grandchild process creation failed!\n");
            exit(1);
        }
        else if (grandpid == 0)
        {
            // Grandchild process
            print_logged_in_users(); // Print logged in users
            exit(0);
        }
        else
        {
            // Child process (after creating grandchild)
            printf("Child process: Grandchild PID = %d\n", grandpid);
            exit(0);
        }
    }
    else
    {
        // Parent process
        wait(NULL); // Wait for child to finish
        printf("Parent process: PID = %d, Child PID = %d\n", getpid(), pid);
        printf("Parent process exiting normally.\n");
        exit(0);
    }

    return 0; // Unreachable code
}
```

OUTPUT:



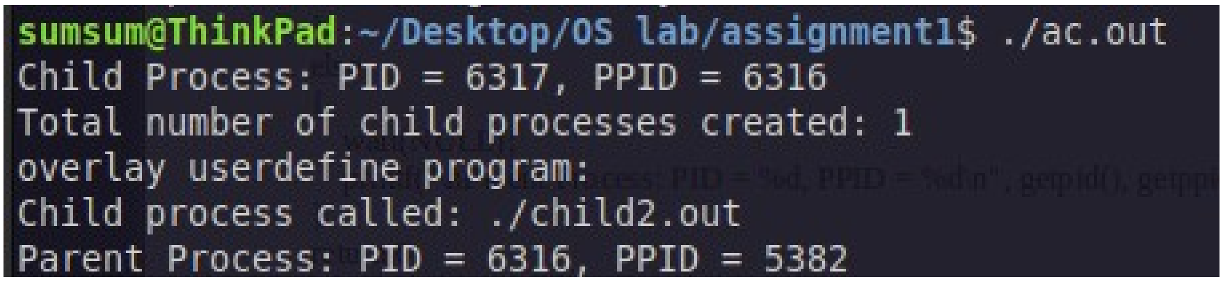
```
sumsum@ThinkPad:~/Desktop/OS lab/assignment1$ ./ab.out
Child process: PID = 6142, PPID = 6141
Child process: Grandchild PID = 6143
Logged in user: sumsum
Parent process: PID = 6141, Child PID = 6142
Parent process exiting normally.
```

1.C) Write a C program like the assignment 1(b) and overlay a user designed program into the address space of the child process using execv() system call. Again use wait() system call to synchronize the execution of parent process in your program until child process finishes. Here use the macro WIFEXITED to capture the returned status of the child in parent process. Also demonstrate the use of argument vector to print the program name by the child process.

CODE:

```c
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>
#include <fcntl.h>
int main()
{
   int cnt_child_p = 0;
   int cnt_grandchild_p = 0;
   pid_t child_p, grandchild_p;
   child_p = fork();
   if (child_p < 0)
   {
      fprintf(stderr, "Fork failed\n");
      return 1;
   }
   else if (child_p == 0)
   {
      cnt_child_p++;
      printf("Child Process: PID = %d, PPID = %d\n", getpid(), getppid());
      printf("Total number of child processes created: %d\n", cnt_child_p);
      char *args[] = {"./child2.out", NULL};
      printf("overlay userdefine program:\n");
      execv(args[0], args);
      perror("execv");
      exit(EXIT_FAILURE);
   }
   else
   {
      wait(NULL);
      printf("\nParent Process: PID = %d, PPID = %d\n", getpid(), getppid());
   }
   return 0;
}
```

OUTPUT:



```
sumsum@ThinkPad:~/Desktop/OS lab/assignment1$ ./ac.out
Child Process: PID = 6317, PPID = 6316
Total number of child processes created: 1
overlay userdefine program:
Child process called: ./child2.out
Parent Process: PID = 6316, PPID = 5382
```