111403003 Akash Sarda
111403019 Jassim Rahman
111403023 Aditya Malu

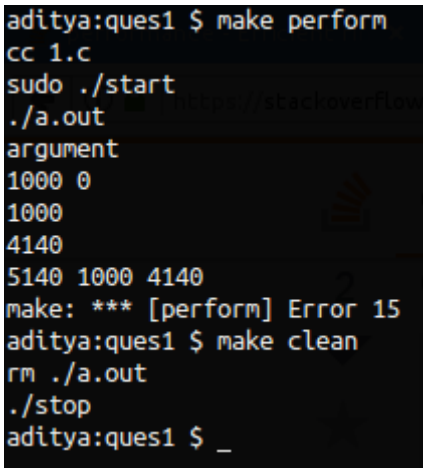
AUP – LAB 8

Write a program to implement the following:

1. Create a new system call wait2, which extends the wait system call.

```
int wait2(int *wtime, int *rtime, int *iotime)
```

Where the three arguments are pointers to integers to which the wait2 function will assign:
a. The aggregated number of clock ticks during which the process waited (was able to run but did not get CPU)
b. The aggregated number of clock ticks during which the process was running
c. The aggregated number of clock ticks during which the process was waiting for I/O (was not able to run).
The wait2 function shall return the pid of the child process caught or -1 upon failure



```
aditya:ques1 $ make perform
cc 1.c
sudo ./start
./a.out
argument
1000 0
1000
4140
5140 1000 4140
make: *** [perform] Error 15
aditya:ques1 $ make clean
rm ./a.out
./stop
aditya:ques1 $ _
```

./start – Starts accounting
./stop – Stops accounting

Code :

```
#include<stdio.h>
#include<unistd.h>
#include <sys/acct.h>
#define ACCFILE "/var/adm/pacct"

FILE *fp;
struct acct acdata;
```

```
unsigned long compt2ulong(comp_t comptime) /* convert comp_t to
unsigned long */
{
     int val; int exp;
     val = comptime & 0x1fff; /* 13-bit fraction */
     exp = (comptime >> 13) & 7; /* 3-bit exponent (0-7) */
     while (exp-- > 0) val *= 8;
     return(val);
}

int wait2(long *rtime,long *wtime){
     int pid;
     long temp;
     pid =  wait(NULL);
     fseek(fp, -sizeof(acdata), SEEK_END);
     fread(&acdata,  sizeof(acdata), 1, fp);
     printf("%ld %ld\n", compt2ulong(acdata.ac_utime),
compt2ulong(acdata.ac_stime));
     temp = compt2ulong(acdata.ac_utime) +
compt2ulong(acdata.ac_stime);
     printf("%ld\n", temp);
     *rtime = temp;

     temp = compt2ulong(acdata.ac_etime - acdata.ac_utime -
acdata.ac_stime);
     //temp = compt2ulong(acdata.ac_etime);
     printf("%ld\n", temp);
     *wtime = temp;
     return pid;
}

int main(){
     long rtime, wtime, iotime, pid;
     fp = fopen("./a", "r");
     if(fork()){
          pid = wait2(&rtime, &wtime);
          printf("%ld %ld %ld\n",pid, rtime, wtime);
     }else{
          int i = 0;
          scanf("%d", &i);
     }
}
```

---

2. Call fork. Let the child create a new session. Verify that the child becomes the process group leader and it does not have a controlling terminal.

```
aditya:lab8 $ gcc 2.c
aditya:lab8 $ ./a.out
In child process...
Before setsid...
Session id: 1, pid: 5360, ppid: 5359 ...
After setsid...
Session id: 5360, pid: 5360, ppid: 5359 ...
The process does not have a controlling terminal...
aditya:lab8 $ _
```

Code:

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

int main(){
     pid_t pid1, pid2, pid;
     int status;
     if((pid1 = fork()) < 0){
          perror("Failed to fork process...\n");
          exit(1);
     }
     if(pid1 == 0){
          printf("In child process...\n");
          printf("Before setsid...\n");
          printf("Session id: %d, pid: %d, ppid: %d ...\n",
getsid(), getpid(), getppid());
          if((pid2 = setsid()) < 0){
               printf("Failed to setsid...\n");
               exit(1);
          } else {
               printf("After setsid...\n");
               printf("Session id: %d, pid: %d, ppid: %d ...\n",
pid2, getpid(), getppid());
          }
          if(open("/dev/tty", O_RDONLY) == -1){
               printf("The process does not have a controlling
terminal...\n");
          } else {
               printf("It has a controlling terminal\n");
          }
     } else {
          pid = wait(&status);
     }
     exit(0);
}
```

---

3. Write a program to verify that a parent process can change the process group ID of one of its children before the child performs an exec(), but not afterward.

```
aditya:lab8 $ gcc 3.c
aditya:lab8 $ ./a.out
Forking child 1
CHILD PID: 5430, PPID: 5429, PGID:5429
Setpgid success
CHILD PID: 5430, PPID: 5429, PGID:5430
Forking child 2
Exiting child 1
CHILD PID: 5431, PPID: 5429, PGID:5429
Setpgid failed: Permission denied
CHILD PID: 5431, PPID: 5429, PGID:5429
Exiting program some...
aditya:lab8 $ _
```

Note: Child 2 execs a simple program called which sleeps for 3 seconds when executed.

Code :

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

int main(){
     pid_t pid1, pid2, pid;
     int status;
     printf("Forking child 1 \n");
     if((pid1 = fork()) < 0){
          perror("Failed to fork process \n");
          exit(1);
     }
     if(pid1 == 0){
          sleep(3);
          printf("Exiting child 1 \n");
          exit(0);
     } else {
          printf("CHILD PID: %d, PPID: %d, PGID:%d \n", pid1,
getpid(), getpgid(pid1));
          if(setpgid(pid1, pid1) == -1){
               perror("Setpgid failed");
          } else {
               printf("Setpgid success \n");
          }
          printf("CHILD PID: %d, PPID: %d, PGID:%d \n", pid1,
```

```
getpid(), getpgid(pid1));
     }

     printf("Forking child 2 \n");
     if((pid2 = fork()) < 0){
          perror("Failed to fork process \n");
          exit(1);
     }
     if(pid2 == 0){
          execl("./some", "some", NULL);
     } else {
          pid = wait(&status);
          printf("CHILD PID: %d, PPID: %d, PGID:%d \n", pid2,
getpid(), getpgid(pid2));
          if(setpgid(pid2, pid2) == -1){
               perror("Setpgid failed");
          } else {
               printf("Setpgid success \n");
          }
          printf("CHILD PID: %d, PPID: %d, PGID:%d \n", pid2,
getpid(), getpgid(pid2));
     }
}
```
