AUP Lab 8
Nikhil Gawande 111408013
Anupam Godse 111408016

Q1. 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

Program:

Statring Accounting:

```c
#include<stdio.h>
#include<unistd.h>

int main(){
    if( acct("./accounting_file") < 0){
        perror("error");
    }
}
```

Stopping Accounting:

```c
#include<stdio.h>
#include<unistd.h>

int main(){
    acct(NULL);
}
```

Main Program:

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

FILE *fp;
struct acct acdata;

unsigned long compt2ulong(comp_t comptime) {
    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((pid = fork()) > 0){
      printf("%ld ",pid);
      pid = wait2(&rtime, &wtime);

      printf("PID:%ld rtime: %ld wtime: %ld\n",pid, rtime, wtime);
  }else{
      int i;
      for(i = 0; i < 5;){
          i++;
      }
      exit(1);
  }
}
```

Make file:

```
run:
	cc a8q1.c
	sudo ./start
	./a.out

clean:
	rm ./a.out
	./stop
```

Outptut:

```
cc a8q1.c
sudo ./start
./a.out
3332 PID:3332 rtime: 1000 wtime: 2332
```

Q2. “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.

Program:

```c
#include<stdio.h>
#include <unistd.h>
#include<stdlib.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
int main(){
    int pid,childPid;
    if((pid = fork()) > 0){
        printf("Child pid: %d\n",pid);
        exit(1);
    }
    else if(pid == 0){
            int fd = open("/dev/tty", O_RDWR);
        if(fd == -1){
            perror("Error ");
            printf("The session doesn't have controlling terminal.\n");
        }
        else{
            printf("control terminal exists before setsid()\n");
        }
        printf("old session id: %d\n",getsid(0));
        printf("new session id: %d\n",setsid());

        close(fd);

        fd = open("/dev/tty", O_RDWR);
        if(fd == -1){
            perror("Error ");
            printf("The session doesn't have controlling terminal after setsid()\n");
        }
    }
    return 0;
}
```

Execution:

```
anupam@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Lab8$ cc a8q2.c -o a
anupam@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Lab8$ ./a
Child pid: 2711
control terminal exists before setsid()
old session id: 2358
new session id: 2711
Error : No such device or address
The session doesn't have controlling terminal after setsid()
anupam@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Lab8$
```

Q3. 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.

Exec Code:

```c
#include<stdio.h>
#include<unistd.h>
#include<stdlib.h>
int main(){
    printf("Hello from EXEC.");
    printf("\n");
    return 0;
}
```

Program:

```c
#define _POSIX_SOURCE
#include<unistd.h>
#include<sys/types.h>
#include<stdio.h>
#include<stdlib.h>
#include <sys/wait.h>
int main() {
  int option;
  pid_t pid;
  printf("Choose one of the options:\n => setpgid() after exec: 1\n => setpgid() before exec: 2\n");
  scanf("%d",&option);
  char *args[]={"./EXEC",NULL};
  switch(option){
    case 1:
        if ((pid = vfork()) == 0) {
          printf("Initially child's PGID is %d\n\n", (int) getpgrp());
          execvp(args[0],args);
        }
        else {
          sleep(2);
          printf("Initially parent's PGID is %d\n\n", (int) getpgrp());
          printf("parent is performing setpgid() on pid %d after exec.\n\n", (int) pid);

          if (setpgid(pid, 0) != 0)
            perror("setpgid() error.");

          printf("PGIDS after performing setpgid() on child:\n");
          printf("parent's PGID is now %d\n", (int) getpgrp());
          printf("child's PGID now %d\n", (int) getpgid(pid));
        }
        break;
    case 2:
        if ((pid = fork()) == 0) {
          sleep(2);
          execvp(args[0],args);
        }
         else {
           printf("Initially child's PGID is %d\n\n", (int) getpgrp());
            printf("Initially parent's PGID is %d\n\n", (int) getpgrp());
           printf("parent is performing setpgid() on pid %d before exec.\n\n", (int) pid);

           if (setpgid(pid, 0) != 0)
             perror("setpgid() error.");

           printf("PGIDS after performing setpgid() on child:\n");
           printf("parent's PGID is now %d\n", (int) getpgrp());
           printf("child's PGID now %d\n", (int) getpgid(pid));
           wait(NULL);
         }
         break;
    }
  return 0;
}
```

Exexution:

```
anupam@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Lab8$ ./a8q3
Choose one of the options:
 => setpgid() after exec: 1
 => setpgid() before exec: 2
2
Initially child's PGID is 3350

Initially parent's PGID is 3350

parent is performing setpgid() on pid 3351 before exec.

PGIDS after performing setpgid() on child:
parent's PGID is now 3350
child's PGID now 3351
Hello from EXEC.
anupam@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Lab8$ ./a8q3
Choose one of the options:
 => setpgid() after exec: 1
 => setpgid() before exec: 2
1
Initially child's PGID is 3352

Hello from EXEC.
Initially parent's PGID is 3352

parent is performing setpgid() on pid 3353 after exec.

setpgid() error.: Permission denied
PGIDS after performing setpgid() on child:
parent's PGID is now 3352
child's PGID now 3352
anupam@vijesh1996-HP-Pavilion-15-Notebook-PC:~/Lab8$
```

Case 1: a parent process can change the process group ID of one of its children before the child performs an exec().

Explaination:

Here, the initial PGIDs of parent and child are: 3350.

Then parent performs setpgid(3351) of child before performing the exec() in child.
Outputs shows that the PGID of child after setpgid(3351) is changed to 3351, this shows that parent can perform setpgid() on child before exec() in child.

Case 2: a parent process can’t change the process group ID of one of its children after the child performs an exec().

Explaination:
Similarly as above case, initial PGIDs of child and parent are: 3352..
Then the child performs exec() after that the parent try to perform setpgid(3353) on child but the setpgid() gives error: Permission Denied this shows that the parent can’t change PGID of child process.