WEEK 6

1. Write a C Program to demonstrate the working of wait and waitpid system calls with a program.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<sys/wait.h>

#include<sys/types.h>

int main()

{

int st;

pid\_t pid1=fork();

pid\_t pid2=fork();

if(pid1==0)

{

printf("first pid:%d\n",getpid());

sleep(2);

exit(0);

}

if(pid2==0)

{

printf("second pid:%d\n",getpid());

sleep(4);

exit(0);

}

wait(&st);

printf("first wait\n");

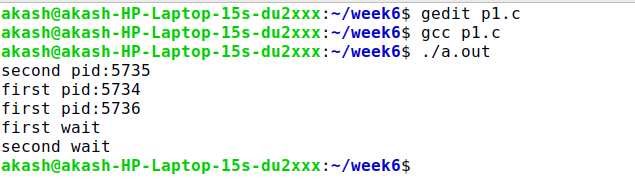
sleep(1);

waitpid(pid2,&st,0);

printf("second wait\n");

return 0;

}



2.Write a program to demonstrate the zombie state of a process and provide the solution for the same.

#include<stdio.h>

#include<stdlib.h>

#include<unistd.h>

#include<fcntl.h>

#include <sys/wait.h>

int main(void) {

pid\_t pid,pid1,pid2;

if ((pid = fork()) < 0) {

printf("fork error");

} else if (pid == 0) { /\* first child \*/

if ((pid1 = fork()) < 0)

printf("fork error");

else if (pid1==0) {

sleep(5);

printf("Child pid is: %d\n",getpid());

printf("second child, parent pid = %ld\n", (long)getppid());

exit(0);

}

else{

printf("Child pid: %d\n",getpid());

exit(0);

}

}

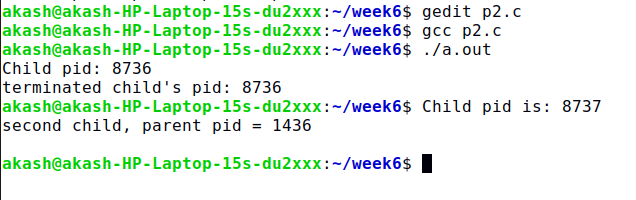
if ((pid2=waitpid(pid, NULL, 0)) != pid)

printf("waitpid error");

printf("terminated child's pid: %d\n",pid2);

exit(0);

}



3. Write a C program to create a child process and show how parent and child processes will share the text file and justify that both parent and child share the same file offset.

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

int main()

{

int fd = open("test.txt", O\_RDWR);

if (fd == -1)

{

perror("open");

return 1;

}

pid\_t pid = fork();

if (pid == -1)

{

perror("fork");

return 1;

}

else if (pid == 0)

{

char buffer[10];

read(fd, buffer, 5);

buffer[5] = '\0';

printf("Child read: %s\n", buffer);

}

else

{

wait(NULL);

char buffer[10];

read(fd, buffer, 5);

buffer[5] = '\0';

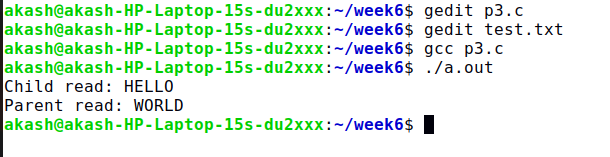
printf("Parent read: %s\n", buffer);

}

close(fd);

return 0;

}



4. Write a C program to demonstrate race condition between parent and child processes

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <fcntl.h>

#include <sys/wait.h>

#include <errno.h>

static void charatatime(char \*);

int main(void)

{

pid\_t pid;

if ((pid = fork()) < 0)

{

printf("fork error");

}

else

if (pid == 0)

{

charatatime("output from child\n");

}

else

{

charatatime("output from parent\n");

}

exit(0);

}

static void charatatime(char \*str)

{

char \*ptr;

int c;

setbuf(stdout, NULL);

/\* set unbuffered \*/

for (ptr = str; (c = \*ptr++) != 0; )

putc(c, stdout);

}

