***Practice Assignment 1***

*A program to get some process related information.*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main()

{

printf("Process ID of current process= %d\n",getpid());

printf("Process ID of parent process= %d\n",getppid());

printf("Real user ID of current process= %d\n",getuid());

printf("Effective user ID of current process= %d\n",geteuid());

printf("Real group ID of current process= %d\n",getgid());

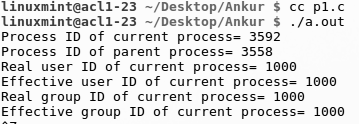
printf("Effective group ID of current process= %d\n",getegid());

for(;;){sleep(1);}

return 0;

}

**Output:**



***Practice Assignment 2***

*A program to create a child process.*

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

int main(void)

{

pid\_t n;

n = fork();

if( n == -1 ) { perror("fork"); exit(1); }

printf("My pid = %u \n", n );

wait(NULL); // This statement would be explained later

return 0;

}

**Output:**



***Practice Assignment 3***

*A program to get the PID of parent and child process.*

#include <unistd.h>

#include <stdlib.h>

#include <stdio.h>

int main( void )

{

pid\_t pid;

pid = fork();

if( pid == -1 ) { perror("fork"); exit(1); }

if( pid == 0 ) // child process

{

exit(0); // child exits

}

// rest is parent process

printf("parent: My PID = %u \n", getpid() );

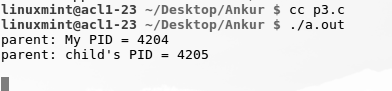
printf("parent: child's PID = %u \n", pid );

while(1) { sleep(1); } // endless loop

return 0;

}

**Output:**



***Practice Assignment 4***

*A program to create multiple child process and the child process may replace itself with another process while maintaining the same process ID.*

#include <sys/types.h>

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <wait.h>

int main(void)

{

pid\_t xx;

int n;

char \*argv[] = { "-d 1", NULL };

// argument list terminated by a NULL pointer. Read execve(2).

char \*argp[] = { "TERM=linux", NULL };

// environment list terminated by a NULL pointer. Read execve(2).

printf( "parent: My pid = %u \n", getpid() );

// main(parent) tries to create a new process

xx = fork();

if( xx == -1 ) { perror("fork-1"); exit(1); }

if ( xx == 0 ) // 1st-child process

{

printf("1st-child: My pid = %u \n", getpid() );

printf("1st-child: My ppid = %u \n", getppid() );

sleep(20);

printf("1st-child: Executing /usr/bin/top %s \n", argv[0]);

sleep(2);

n = execve ("/usr/bin/top", argv, argp);

if( n == -1 ) { perror("execve"); }

// [execve] never returns if successful

printf("[execve] ERROR!\n");

exit(2);

}

// parent process

// parent again tries to create a new process

xx = fork();

if( xx == -1 ) { perror("fork-2"); exit(3); }

if ( xx == 0 ) // 2nd-child process

{

printf( "2nd-child: My pid = %d \n",getpid() );

printf( "2nd-child: My ppid = %d \n",getppid() );

printf( "2nd-child: Executing shell script [./s1]\n" );

n = execl("./s1", NULL );

if( n == -1 ) { perror("execl"); }

// [execl] never returns if successful

printf("[execl ERROR !\n");

exit(4);

}

//parent process

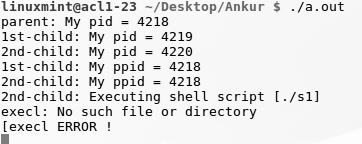
wait(NULL);

wait(NULL);

return 0;

}

**Output:**



***Innovative Assignment***

***Write a “Hello world” greetings program and compile it. Now write another program to create a child process which executes your hello world program.***

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h> /\* for fork \*/

#include <sys/types.h> /\* for pid\_t \*/

#include <sys/wait.h> /\* for wait \*/

int main(int argv , char \* argc[])

{

//printf("%s",argc[2]);

/\*Spawn a child to run the program.\*/

pid\_t pid=fork();

if (pid==0) { /\* child process \*/

execv(argc[2],NULL);

exit(127); /\* only if execv fails \*/

}

else { /\* pid!=0; parent process \*/

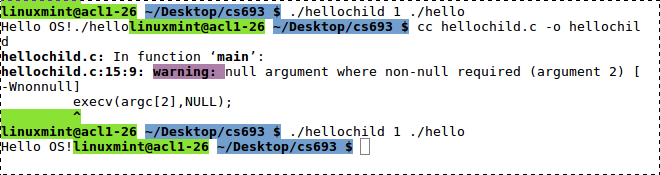
waitpid(pid,0,0); /\* wait for child to exit \*/

}

return 0;

}

**Output :**

****