**Name- Manish kumar**

**Roll- 2020ITB007**

**Group- HX**

**Assignment – 2**

**Operating System Lab**

**1. Write a program showing that the parent and child process will not**

**share the memory location, even global memory. How do we create a child**

**process so that the child and parent share their memory?**

#include <stdio.h>

#include <unistd.h>

int x = 2;

int main()

{

int y = 2;

if (fork() == 0)

{

printf("Child process \n x = %d \n y = %d \n", x++, y++);

}

else

{

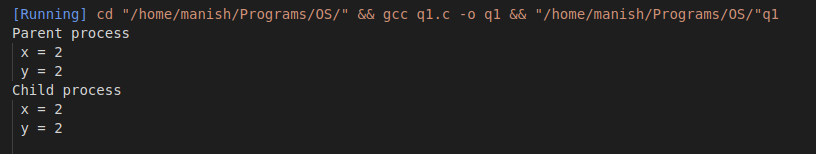
printf("Parent process \n x = %d \n y = %d \n", x--, y--);

}

return 0;

}

**Output :**

****

**2. Write a program that creates a chain of n processes, where n is a**

**command–line argument.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

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

{

int i, n;

n = atoi(argv[1]);

for (i = 0; i < n; i++)

{

if (fork() == 0)

{

printf("Child \n pid = %d \t ppid = %d \n", getpid(), getppid());

}

else

{

printf("Parent \n pid = %d \t ppid = %d \n", getpid(), getppid());

break;

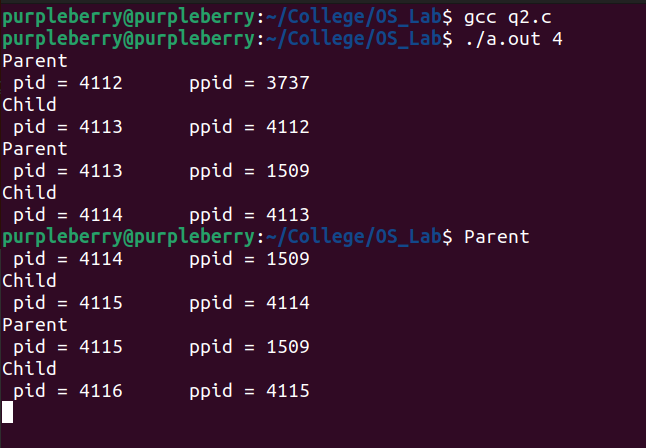
}

}

return 0;

}

**Output :**

****

**3. Write a program that creates a tree of n processes where n is passed**

**as a command-line argument. Then the original process should print the**

**information after all children have exited.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

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

{

int i;

for (i = 0; i < atoi(argv[1]); i++)

{

if (fork() == 0)

{

printf("Child %d \n pid = %d \t ppid = %d \n", i + 1, getpid(), getppid());

exit(0);

}

}

for (i = 0; i < atoi(argv[1]); i++)

{

wait(NULL);

}

printf("\nParent \n pid = %d \t ppid = %d \n", getpid(), getppid());

return 0;

}

**Output :**



**4. Write a program that creates a Zombie process. The program will show**

**the process table entry to ensure that the child process becomes Zombie.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

if (fork() == 0)

{

printf("Child \n pid = %d \t ppid = %d \n\n", getpid(), getppid());

exit(0);

}

else

{

sleep(5);

printf("Parent \n pid = %d \t ppid = %d \n\n", getpid(), getppid());

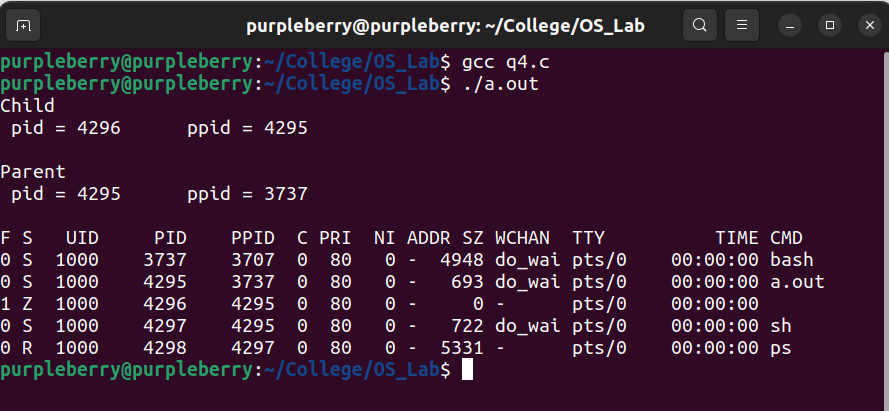
system("ps -l");

}

return 0;

}

**Output :**



**5. Write a C program where the parent must wait for the child to die. And**

**then show the exit status of the child process.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <sys/types.h>

#include <sys/wait.h>

int main()

{

int status;

if (fork() == 0)

{

printf("Child \n pid = %d \t ppid = %d \n\n", getpid(), getppid());

}

else

{

wait(&status);

printf("Parent \n pid = %d \t ppid = %d \n\n", getpid(), getppid());

printf("Child status = %d \n\n", WIFSIGNALED(status));

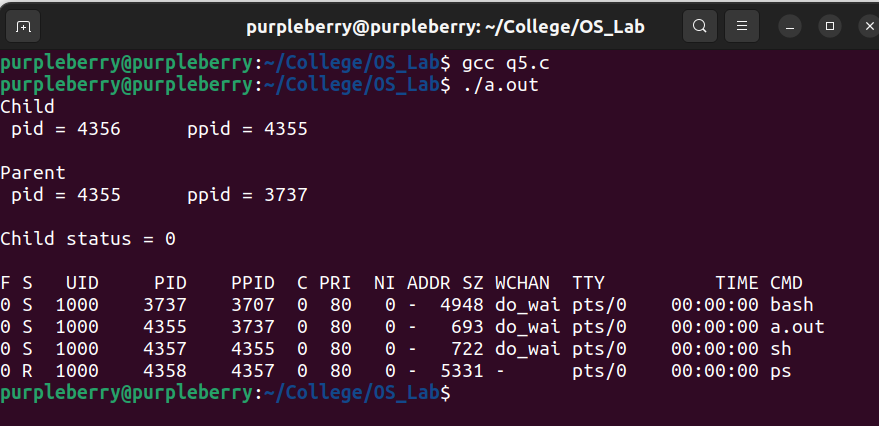
system("ps -l");

}

return 0;

}

**Output :**

****

**6. Write a program that creates a child process that will replace itself with**

**the existing system code of “ls” let show.**

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

int main()

{

if (fork() == 0)

{

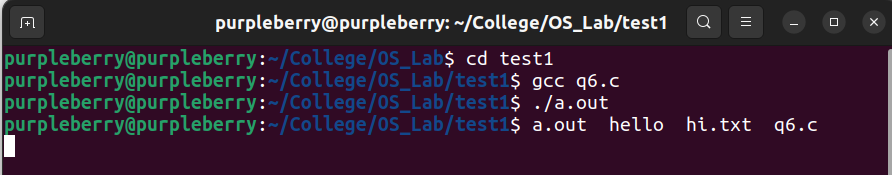
execlp("ls", "ls", NULL);

}

return 0;

}

**Output :**



**7. Write a program that creates a child process to execute a command.**

**The command to be executed is passed on the command line.**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

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

{

if (fork() == 0)

{

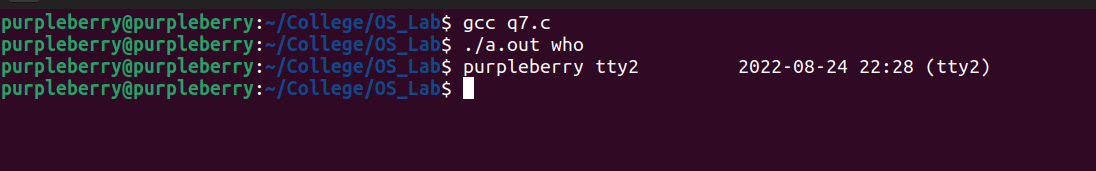
execvp(argv[1], argv + 1);

}

return 0;

}

**Output :**

****

**8. Write a menu-driven program which will display every UNIX command**

**and run on the user’s choice.**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <sys/types.h>

#include <sys/wait.h>

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

{

int ch;

char \*command;

printf("1. ls \n");

printf("2. who \n");

printf("3. pwd \n");

printf("\nEnter your choice : ");

scanf("%d", &ch);

switch (ch)

{

case 1:

command = "ls";

break;

case 2:

command = "who";

break;

case 3:

command = "pwd";

break;

default:

printf("Invalid choice.\n");

}

if (fork() == 0)

{

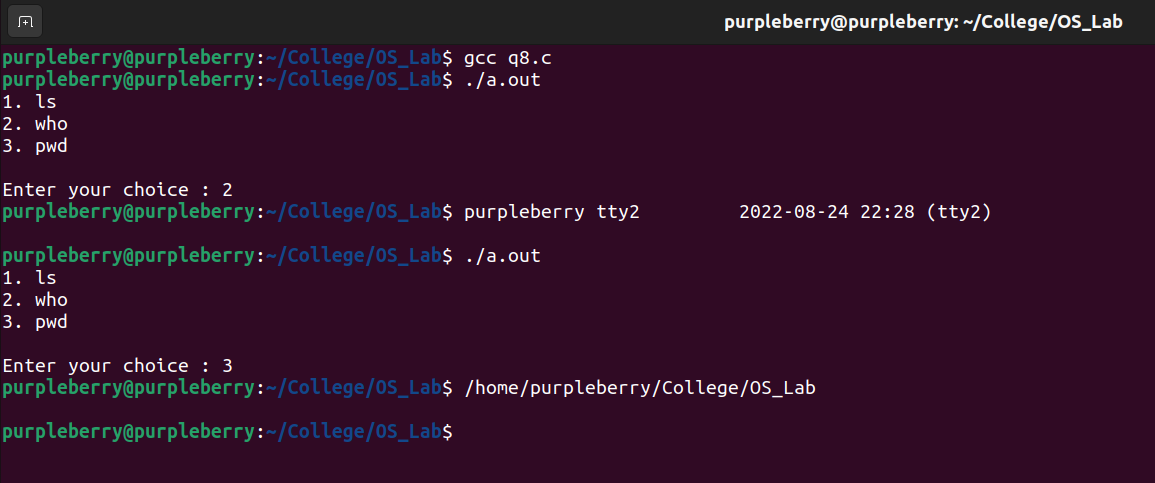
execlp(command, command, NULL);

}

return 0;

}

**Output :**

****