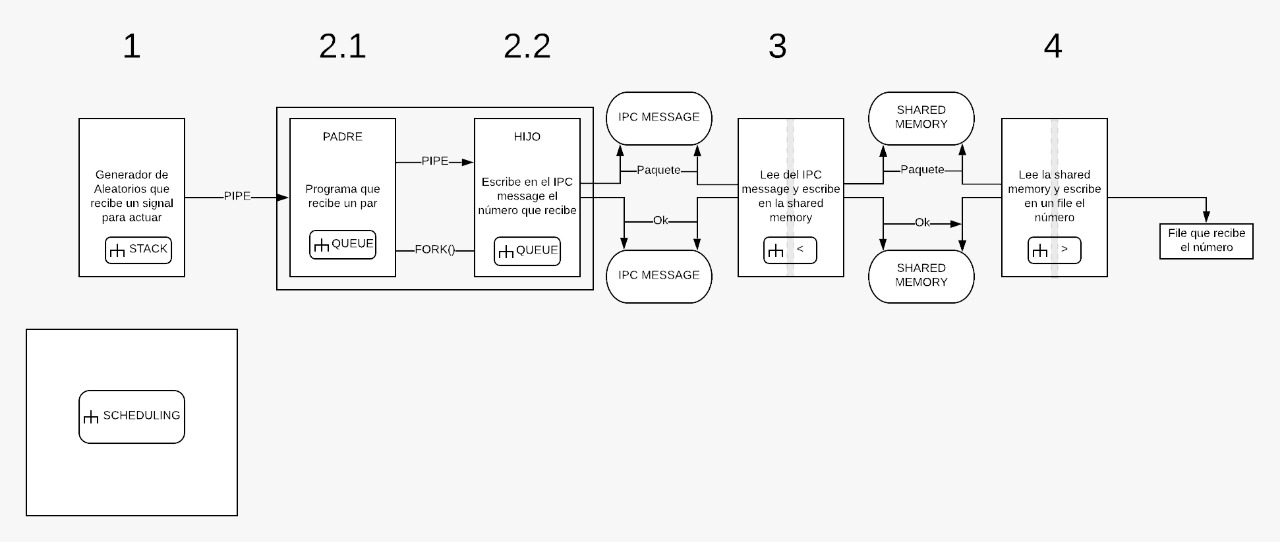
**Informe Proyecto Final**

Integrantes:

* Alonso Gonzalo Valdivia Quispe
* Oscar Daniel Ramos Ramirez
* Fabrizio Rodrigo Flores Pari

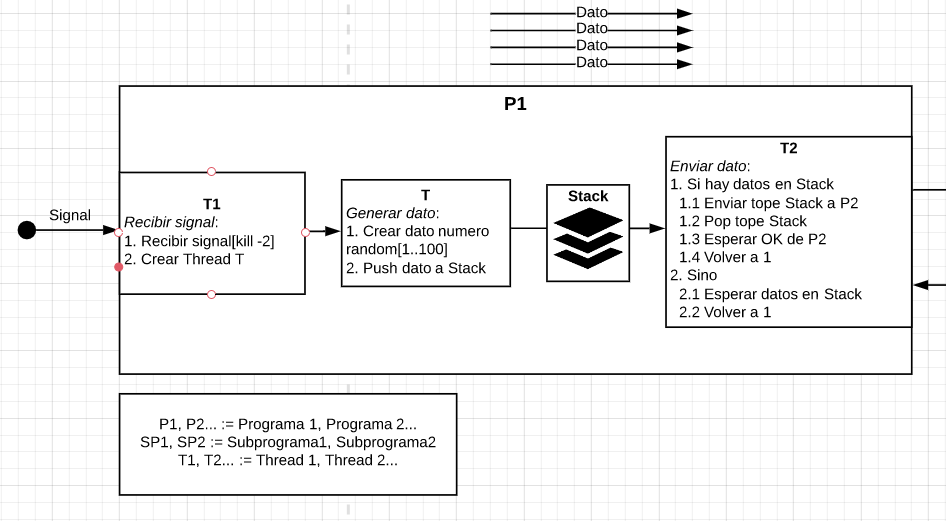
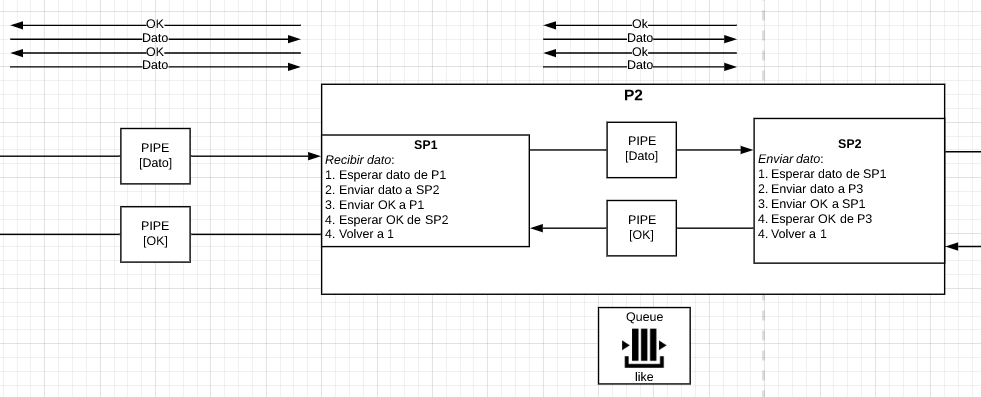
1. **Diagrama/Flujo del proyecto:**

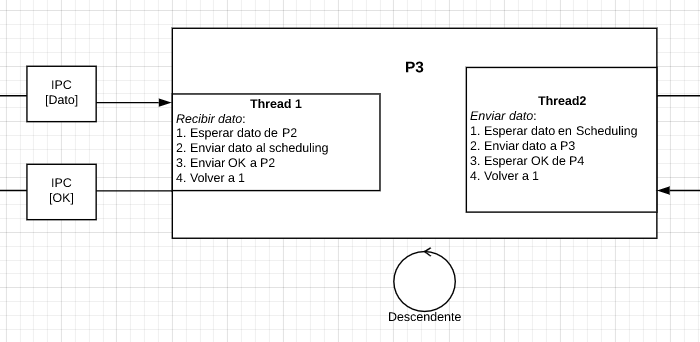
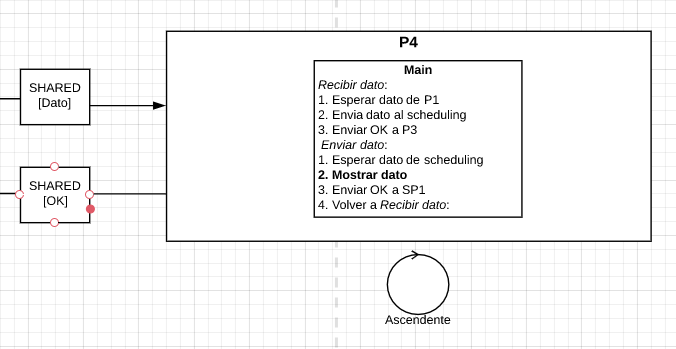


1. **Descripción de la idea**

La idea del proyecto es trasladar un mensaje , en este caso números generados aleatoriamente, mediante diferentes mecanismos de comunicación entre procesos(PIPE, IPC MESSAGE, SHARED MEMORY) con sus respectivos schedulings en cada etapa (STACK, QUEUE, SORT ASCENDENTE Y DESCENDENTE)para observar el comportamiento de este mensaje por el flujo del proyecto.

1. **Especificación de los programas**





1. **Que problema se tuvo al implementar los programas**

* Un problema que nos encontramos al realizar este proyecto fue en recibir una gran cantidad de signal al mismo tiempo.
* Muchos de estos se pierden porque el programa no es lo suficientemente rápido para recibirlos todos, intentamos solucionar este problema usando el comando “nice” sobre el programa pero este problema seguía sin solucionarse.
* Nos planteamos que no solo se envíen signals del tipo 2 sino de varios tipos repartiendo el trabajo entre diversos tipos de signal de tal manera que la pérdida de signal se reduzca al mínimo.
* Pero vimos por conveniente no hacer esta implementación de esta posible solución ya que era muy compleja.
* Concluimos que la forma más simple de tratar este problema era usando sleep después de enviar un signal.

1. **Codigo**

* **Programa 1:**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <queue>

#include <list>

#include <thread>

#include <mutex>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX\_BUF 1024

#define DATA\_SHM\_KEY 54321

#define OK\_SHM\_KEY 98765

#define SHMSZ 1024

using namespace std;

//////////////////////////SCHEDULING//////////////////////

list<int> scheduling;

/////////////////////////////////////////////////////////

//////////////////////SHM////////////////////

int createSHM(int shm\_key)

{

int shmid;

int msgflg = 0666;

size\_t size = SHMSZ;

key\_t key = shm\_key;

if((shmid = shmget(key,size,msgflg)) <= 0)

printf("Fallo al crear SHM[%d], es posible que este ocupado\n",shm\_key);

else

printf("Conexion SHM[%d] establecida\n",shm\_key);

return shmid;

}

void mostrarData();

void AdquirirData(char \*&shmptr)

{

char \*sptr = shmptr;

scheduling.push\_back(atoi(sptr));

scheduling.sort();

scheduling.reverse();

\*shmptr='\*';

mostrarData();

}

void mostrarData()

{

int data = \*(scheduling.begin());

scheduling.pop\_front();

printf("El dato enviado es %d\n",data);

}

void reciveData(char \*&shmptr)

{

while(\*shmptr=='\*');

AdquirirData(shmptr);

}

void sendOkSHM(char data[], char \*&shmptr)

{

printf("Enviaremos el OK\n");

char \*sptr = shmptr;

int c;

for(c = 0; c < strlen(data); c++)

\*sptr++ = data[c];///Enviando el dato SHM

printf("Ok enviado\n");

}

/////////////////////////////////////////////////////////////////

int main()

{

int shmidData = createSHM(DATA\_SHM\_KEY);

int shmidOk = createSHM(OK\_SHM\_KEY);

char \*sdata = (char \*)shmat(shmidData,NULL,0);

char \*sok = (char \*)shmat(shmidOk,NULL,0);

while(1)

{

reciveData(sdata);

sendOkSHM("Ok",sok);

}

return 0;

}

* **Programa 2**

#include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <queue>

#include <thread>

#include <mutex>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define MAX\_BUF 1024

#define PIPE\_P1toP2 "/tmp/pipe\_P1toP2"

#define PIPE\_P2toP1 "/tmp/pipe\_P2toP1"

#define IPC\_KEY 12321

using namespace std;

////////////////////////Pipe////////////////////////

int createPipe(const char\* pipe\_dir, int flag) // Recibe, lee

{

// printf("Pipe [%s] flag [%d]\n", pipe\_dir, flag);

int fd;

mkfifo(pipe\_dir, 0666);

fd = open(pipe\_dir, flag);

// printf("Terminando pipe\n");

return fd;

}

////////////////////////IPC////////////////////////////

/\*

struct msgbuf

{

long mtype;

char mtext[MAXBUF];

};

\*/

int createIPC(int ipc\_key) // Retorna msgid

{

int msgid;

int msgflg = IPC\_CREAT | 0666;

key\_t key = ipc\_key;

if ((msgid = msgget(key, msgflg )) <= 0)

printf("Fallo al crear ipc[%d]!!\n", ipc\_key);

else

printf("Conexion IPC[%d] establecida\n",ipc\_key);

return msgid;

}

int cnt\_send = 1;

void sendDataIPC(int msgid, char data[])

{

struct msgbuf sbuf;

sbuf.mtype = 1;

strcpy(sbuf.mtext, data);

int buflen = strlen(sbuf.mtext)+1;

if (!(msgsnd(msgid, &sbuf, buflen, IPC\_NOWAIT) < 0))

printf("[%d]Enviado %d\n", cnt\_send++, atoi(data));

else

printf ("Error, mensaje IPC no enviado!!: %d, %ld, %s, %d \n", msgid, sbuf.mtype, sbuf.mtext, (int)buflen); // Puede que este lleno el ipc

}

////////////////////////SubProcess////////////////////////////

int fd\_sp12[2]; // fd del SubProceso 1 al 2 [dato]

int fd\_sp21[2]; // fd del SubProceso 2 al 1 [ok]

int cnt\_rec = 1;

// Sub process 1

void receiverData()

{

char buf[MAX\_BUF];

close(fd\_sp12[0]); // Cierro in

close(fd\_sp21[1]); // Cierro out

int fd\_12, fd\_21;

printf("Creacion pipes\n");

fd\_12 = createPipe(PIPE\_P1toP2, O\_RDONLY);

fd\_21 = createPipe(PIPE\_P2toP1, O\_WRONLY);

printf("Conexion Pipes establecida\n");

while(true)

{

read(fd\_12, buf, MAX\_BUF); // Espero de pipe

printf("[%d]Recibo %s\n", cnt\_rec++, buf);

write(fd\_sp12[1], buf, strlen(buf)+1); // envio Dato a SP2

write(fd\_21, "OK", strlen("OK")+1); // envio OK a P1

read(fd\_sp21[0], buf, MAX\_BUF); // espero ok de SP2

}

}

// Sub process 2

void processData()

{

char buf[MAX\_BUF];

close(fd\_sp12[1]); // Cierro out

close(fd\_sp21[0]); // Cierro in

int msgid = createIPC(IPC\_KEY);

while(true)

{

read(fd\_sp12[0], buf, MAX\_BUF); // Espero y obtengo dato

sendDataIPC(msgid, buf);

write(fd\_sp21[1], "OK", strlen("OK")+1); // Envio OK a SP1

}

}

//////////////////////////////Main//////////////////////////////

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

{

printf("PID = %d\n", getpid());

pipe(fd\_sp12);

pipe(fd\_sp21);

int pid = fork();

if(pid == 0) // child, SP1

receiverData();

else

processData();// SP2

return 0;

}

* **Programa 3**

**#**include <stdio.h>

#include <unistd.h>

#include <stdlib.h>

#include <string.h>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/shm.h>

#include <sys/sem.h>

#include <signal.h>

#include <fcntl.h>

#include <sys/stat.h>

#include <sys/types.h>

#include <queue>

#include <list>

#include <thread>

#include <mutex>

#include <sys/types.h>

#include <sys/ipc.h>

#include <sys/msg.h>

#define SHMSZ 1024

#define MAX\_BUF 1024

#define DATA\_IPC\_KEY 12321

#define OK\_IPC\_KEY 56789

#define DATA\_SHM\_KEY 54321

#define OK\_SHM\_KEY 98765

using namespace std;

list<int> scheduling;

mutex mtx;

///////////////////////SCHEDULING/////////////////////////

void pushData(int data)

{

mtx.lock();

scheduling.push\_back(data);

scheduling.sort();//Se ordena de menor a mayor por defecto

mtx.unlock();

}

bool isEmpty()

{

mtx.lock();

bool ans = scheduling.empty();

mtx.unlock();

return ans;

}

int popTop()

{

mtx.lock();

int top = \*(scheduling.begin());

scheduling.pop\_front();

mtx.unlock();

return top;

}

///////////////////////SHARED MEMORY//////////////////////

int createSHM(int shm\_key)

{

int shmid;

int msgflg = IPC\_CREAT | 0666;

size\_t size = SHMSZ;

key\_t key = shm\_key;

if((shmid = shmget(key,size,msgflg)) <= 0)

printf("Fallo al crear SHM[%d], es posible que este ocupado\n", shm\_key);

else

printf("Conexion SHM[%d] establecida\n",shm\_key);

return shmid;

}

int cnt\_send = 1;

void sendDataSHM(int data, char \*shmptr)

{

char buf[MAX\_BUF];

sprintf(buf,"%d",data);

char \*sptr = shmptr;

struct msgbuf sbuf;

sbuf.mtype = 1;

int c;

printf("Se enviara el dato %s\n",buf);

for(c = 0; c < strlen(buf); c++)

\*sptr++ = buf[c];///Enviando el dato SHM

\*sptr = '\n';

}

void waitOKSHM(char \*shmptr)

{

char \*rptr = shmptr;

while(\*rptr == '\*')

{

// printf("Esperando OK\n");

// sleep(1);

}

// printf("Ok recibido\n");

\*shmptr='\*';

}

////////////////////////IPC///////////////////////////////

int createIPC(int ipc\_key) // Retorna msgid

{

int msgid;

int msgflg = IPC\_CREAT | 0666;

key\_t key = ipc\_key;

if ((msgid = msgget(key, msgflg )) <= 0)

printf("Fallo al crear ipc[%d]!!\n", ipc\_key);

else

printf("Conexion IPC[%d] establecida\n",ipc\_key);

return msgid;

}

void sendDataIPC(int msgid, char data[])

{

struct msgbuf sbuf;

sbuf.mtype = 1;

strcpy(sbuf.mtext, data);

int buflen = strlen(sbuf.mtext)+1;

if (!(msgsnd(msgid, &sbuf, buflen, IPC\_NOWAIT) < 0))

printf("[%d]Enviado %d\n", cnt\_send++, atoi(data));

else

printf ("Error, mensaje IPC no enviado!!\n"); // Puede que este lleno el ipc

}

void reciveData(int msgid,int msgidOk)

{

struct msgbuf rbuf;

rbuf.mtype = 2;

while(1)

{

if((msgrcv(msgid,&rbuf, MAX\_BUF,1,0)) < 0)

printf("El mensaje no se recibio %s\n",rbuf.mtext);

else

printf("Se recibio el mensaje %s\n",rbuf.mtext);

pushData(atoi(rbuf.mtext));

sendDataIPC(msgidOk,"Ok");

}

}

//////////////////////////Envio y Recepcion de Datos///////////////////////////////

void RecivePaquetes()

{

// printf("Empieza Thread 2\n");

int ipcRcv = createIPC(DATA\_IPC\_KEY);

int ipcOk = createIPC(OK\_IPC\_KEY);

reciveData(ipcRcv,ipcOk);

}

void SendPaquetes()

{

// printf("Empieza Thread 1\n");

int shmidData = createSHM(DATA\_SHM\_KEY);

int shmidOk = createSHM(OK\_SHM\_KEY);

char \*sdata = (char \*)shmat(shmidData,NULL,0);

char \*sok = (char \*)shmat(shmidOk,NULL,0);

while(1)

{

if(!isEmpty())

{

int data = popTop();

printf("Se envia paquete %d\n",data);

sendDataSHM(data,sdata);

waitOKSHM(sok);//Aquí esperar

}

else

{

while(isEmpty());

}

}

}

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

{

thread t2(SendPaquetes);

thread t3(RecivePaquetes);

// printf("Todo bien\n");

t2.join();

t3.join();

return 0;

}

* **Programa 4**