EstructurasPrehechasFinal.pdf

liebana147776

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Técnica Superior de Ingeniería Industrial, Informática y de Telecomunicación
Universidad Pública de Navarra

Accede al documento original


```
//MAIN CON ARGUMENTOS
int main(int argc, char **argv){
   return 0;
}

// CREAR PROCESO
{
   pid_t pidHijo = fork();

   if(pidHijo == 0){ //HIJO
   }else if(pidHijo > 0){ //PADRE
      waitpid(pidHijo,0,0);
   }else{
      perror("Error en la creacion de procesos.\n");
      exit(1);
   }
}

//NIETO
{
   pid_t pidNieto;
   pidNieto = fork();
   if (pidNieto==0){ //NIETO
   }else if(pidNieto > 0){ //HIJO
      waitpid(pidNieto,0,0);
   }else{
      perror("Error en la creacion de procesos.\n");
      exit(1);
   }
}
```

```
//TUBERIA
{
   //CREAR TUBERIA O PIPE
   int tuberia[2];
   pipe(tuberia);
   //CERRAR TUBERIA
   close(tuberia[1]);
   close(tuberia[0]);
}

//PROCESOS COMUNICADOS POR TUBERIA
{
   typedef struct mensaje {
   }mensaje;

   int main(){
      int pipe_a[2];
      pipe(pipe_a);

      int pipe_b[2];
      pipe(pipe_b);

      pid_t pid = fork();

      if(pid == 0){ //Hijo
         close(pipe_a[1]);
         close(pipe_b[0]);

         read(pipe_a[0],&msg,sizeof(msg));  //LEO DE LA PIPE
         write(pipe_b[1],&msg,sizeof(msg));//ESCRIBO  EN LA PIPE
```

```
        close(pipe_a[0]);
        close(pipe_b[1]);
    }else if(pid > 0){ //Padre
        close(pipe_a[0]);
        close(pipe_b[1]);

        read(pipe_b[0],&msg,sizeof(msg));  //LEO DE LA PIPE
        write(pipe_a[1],&msg,sizeof(msg));//ESCRIBO  EN LA PIPE

        close(pipe_a[1]);
        close(pipe_b[0]);

        waitpid(pid,0,0);
    }else{ //Error
        printf("Error en la creacion de proceso\n");
        exit(1);
    }
    return 0;
  }
}

//FIFO ENVIAR
{
  #define NOMBREFIFO "mififo_enviar"
  #define NOMBREFIFO2 "mififo_recibir"

  typedef struct mensaje {
  }mensaje;

  int main() {
    int fp_e, fp_r;
```


```
    // Crear los FIFO si no existen
    mkfifo(NOMBREFIFO, S_IFIFO | 0660); //Crear FIFO
    mkfifo(NOMBREFIFO2, S_IFIFO | 0660);

    // Abrir FIFO de escritura y lectura
    fp_e = open(NOMBREFIFO, O_WRONLY); // Abrir FIFO de escritura
    fp_r = open(NOMBREFIFO2, O_RDONLY);// Abrir FIFO de lectura

    write(fp_e, &msg, sizeof(mensaje));
    read(fp_r, &msg, sizeof(mensaje));

    close(fp_e);
    close(fp_r);

    return 0;
  }
}

//FIFO RECIBIR
{
  #define NOMBREFIFO "mififo_enviar"
  #define NOMBREFIFO2 "mififo_recibir"

  typedef struct mensaje {
  } mensaje;

  int main() {
    int fp_e, fp_r;
    mensaje msg;
```

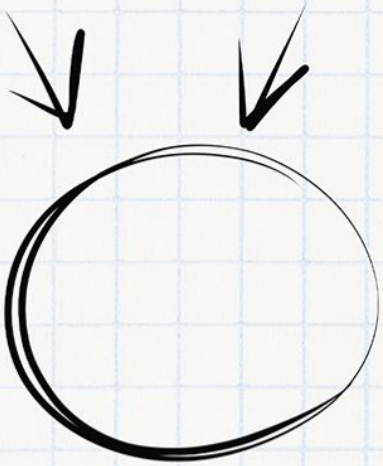
```
    fp_e = open(NOMBREFIFO, O_RDONLY); // Abrir FIFO de lectura
    fp_r = open(NOMBREFIFO2, O_WRONLY);// Abrir FIFO de escritura

    read(fp_e, &msg, sizeof(mensaje));
    write(fp_r, &msg, sizeof(mensaje));

    close(fp_e);
    close(fp_r);

    return 0;
  }
}

//SEÑALES
{
  void signalHandler(int e){} //Manejador de señal
  signal(SEÑAL,signalHandler);//Preparacion de señal
  kill(PID,SEÑAL);          //Enviar señal

  SIGHUP (1): Cuelga el proceso
  SIGINT (2): Interrumpe el proceso
  SIGQUIT (3): Termina el proceso
  SIGILL (4): Instrucción ilegal
  SIGTRAP (5): Trampa de rastreo
  SIGABRT (6): Abortar
  SIGBUS (7): Error de bus
  SIGFPE (8): Error de punto flotante
  SIGKILL (9): Matar el proceso
  SIGUSR1 (10): Señal definida por el usuario 1
  SIGSEGV (11): Violación de segmento
  SIGUSR2 (12): Señal definida por el usuario 2
```

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
Descargas sin publi al mes	10	40	80
Elimina el video entre descargas	✓	✓	✓
Descarga carpetas	✗	✓	✓
Descarga archivos grandes	✗	✓	✓
Visualiza apuntes online sin publi	✗	✓	✓
Elimina toda la publi web	✗	✗	✓
Precios Anual	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo, ¿Qué nota vas a sacar?



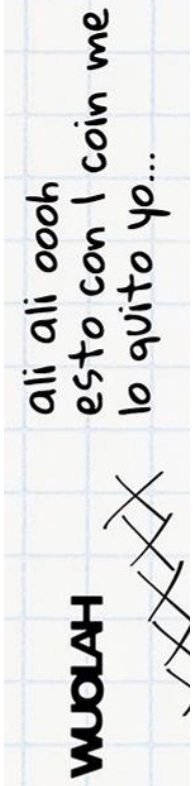
WUOLAH

```
   SIGPIPE (13): Escribe en una tubería sin lectores
   SIGALRM (14): Alarma de reloj
   SIGTERM (15): Terminación del proceso
   SIGCHLD (17): Cambio de estado del hijo
   SIGCONT (18): Continuar si se detiene
   SIGSTOP (19): Detener el proceso
}
//ACABAR ORDENADAMENTE
{
   pid_t pidHijo;

   void cntrl_c(int e){
      kill(pidHijo,SIGINT);       //CUIDADO SI HAY NIETOS *1
      waitpid(pidHijo,0,0);

      exit(0);
   }
}

//REDIRIGIR ENTRADA/SALIDA
{
   void prep_redirection(char**  arg,int i, int flags, int inUoutFILENO){
      int fd = open(arg[i], flags,0600);
      dup2(fd,inUoutFILENO);
      close(fd);
   }

   prep_redirection(NOMBREFICHERO,i,  FLAGS, STDIN_FILENO); //REDIRIGIR ENTRADA
   prep_redirection(NOMBREFICHERO,i,  FLAGS, STDOUT_FILENO); //REDIRIGIR SALIDA

   FILE* input;
```


```
   input = stdin; //Redirigir entrada estandar(TECLADO)
   input = stdout; //Redirigir salida estandar(TERMINAL)

   fscanf(input,"%ld %d %[^\n]",&process.level,&process.priority,process.command)) !=
EOF //LEER del fichero "input"

   /*
    *
    * ENTRADA ESTANDAR -> stdin
    *     scanf -> stdin
    *     fscanf(stdin, ...)
    *
    * SALIDA ESTANDAR -> stdout
    *     printf -> stdout
    *     fprintf(stdout,...)
    *
    * Sin embargo podemos redirigir antes con dup2(fd,STDIN/STDOUT)
    * y luego hacer el printf
    *
    * Si hemos redirigido la entrada y queremos printear por terminal
    * usaremos:
    *     fprintf(stderr,..)
    */
}

//SEMAFOROS
{
   //Crear grupo semaforico
   crearSemaforo(clave,numSemaforos,color);

   // operar semaforo a verde
```

```
operarSemaforo( semid, color, numSem);

// operar semaforo a rojo
operarSemaforo(semid, (-1)*color, numSem);

//Borrar semaforo
semctl(semid, 0, IPC_RMID);

//Si sem_op es un número entero negativo: Si el valor absoluto de sem_op es menor o igual que el valor del semáforo (semval), entonces se resta sem_op de semval.
// Si el valor absoluto de sem_op es mayor que semval, entonces el proceso se bloquea hasta que semval sea al menos el valor absoluto de sem_op.

//Si sem_op es un número entero positivo: Se suma sem_op a semval y se despiertan todos los procesos que estaban esperando (bloqueados) en ese semáforo.

//Si sem_op es cero: Si semval es cero, la operación puede continuar. Si semval no es cero, entonces el proceso se bloquea hasta que semval sea cero.
//semop()
semop(semid, &accion, 1); // el 1, es el número de semaforos en el grupo semaforico

//sem_num que es el índice del array del semáforo sobre el que queremos actuar. En nuestro caso, con un sólo semáforo, el índice será 0.
//sem_op que es el valor en el que queremos decrementar el semáforo. En nuestro caso, -1.
//sem_flg son flags que afectan a la operación. En nuestro caso, para no complicarnos la vida, pondremos 0.

//Al realizar esta operación, si el semáforo se vuelve negativo, nuestro proceso se quedará "bloqueado" hasta que alguien incremente el semáforo y lo haga, como mínimo, 0.

//Crear grupo semaforico
union semun {
```

```
    int val;
    struct semid_ds *buf;
    ushort *array;
};

int semid;
void crearSemaforo(int clave,int numSemaforos,int color){
    union semun arg;
    key_t key = ftok("/tmp", clave);
    if(key == -1){
        perror("[!] Error al hacer la creacion de la KEY");
        exit(-1);
    }
    // creación del semaforo
    semid = semget(key, numSemaforos, 0660 | IPC_CREAT);
    if(semid == -1){
        perror("[!] Error al hacer la CREACION del grupo semaforico");
        exit(-1);
    }

    // inicializar semaforo
    arg.val = color; // <= 0 Rojo a esa cantidad, >0 Verde a esa cantidad
    if(semctl(semid, 0, SETVAL, arg) == -1){
        perror("[!] Error al hacer la INICIALIZACION del grupo semaforico");
        exit(-1);
    }
}
//OPERAR-SEMAFORO
void operarSemaforo(int semid,int color, int numSem) {
    struct sembuf sops; //Signal
    sops.sem_num = numSem;
```




```
   sops.sem_op = color;
   sops.sem_flg = 0;

   if (semop(semid, &sops, 1) == -1) {
      perror("[!] Error al hacer el SIGNAL");
      exit(-1);
   }
}
}

//MENSAJES
{
   key_t key = ftok("/tmp",clave);

   int msgrid = msgget(key,0666 | IPC_CREAT); //Crear cola de mensajes

   msgrid = msgget(key,0666 | IPC_CREAT); //Enlazar cola de mensajes

   msgsnd(msgrid,&msg,sizeof(msg) - sizeof(long),0); //Mandar mensaje BLOQUEANTE

   msgsnd(msgrid,&msg,sizeof(msg) - sizeof(long),IPC_NOWAIT); //Mandar mensaje NO
BLOQUEANTE

   msgrcv(msgrid,&msg,sizeof(msg) - sizeof(long),1,0); //Recibir mensaje BLOQUEANTE

   msgrcv(msgrid,&msg,sizeof(msg) - sizeof(long),1,IPC_NOWAIT); //Recibir mensaje NO
BLOQUEANTE

   msgctl(msgrid, IPC_RMID, NULL); //Borrar cola de mensajes

}
```

```
//makefile
{
   #makefile

   all: cola fragmenta main

   cola: colaEntero.c
      gcc -c colaEntero.c -Wall
   fragmenta: fragmenta.c
      gcc -c fragmenta.c -Wall
   main: main.c colaEntero.o
      gcc main.c fragmenta.o colaEntero.o -o main
}
```