



UNIVERSIDAD NACIONAL DE LA MATANZA

Departamento de Ingeniería e Investigaciones Tecnológicas

Sistemas Operativos (Plan 2009)

Jefe de Cátedra: Fabio E. Rivalta

Equipo de Docentes: Boettner F., Catalano L., de Lizarralde R, Villamayor A.

Auxiliares docentes: Loiacono F., Hirschfeldt D., Piubel F., Rodriguez A., Segura L., Fernandez Piñeiro, Radice A.

C y Multi-procesos

SISTEMAS OPERATIVOS UNLAM.

OBJETIVOS

- Repasar compilación en C
- Programar con multiprocesos
- Sincronizar procesos
- Comunicar procesos



ROADMAP

- Compilación
- Procesos Pesados
- Procesos Livianos
- Semáforos
- PIPE
- FIFO
- Socket



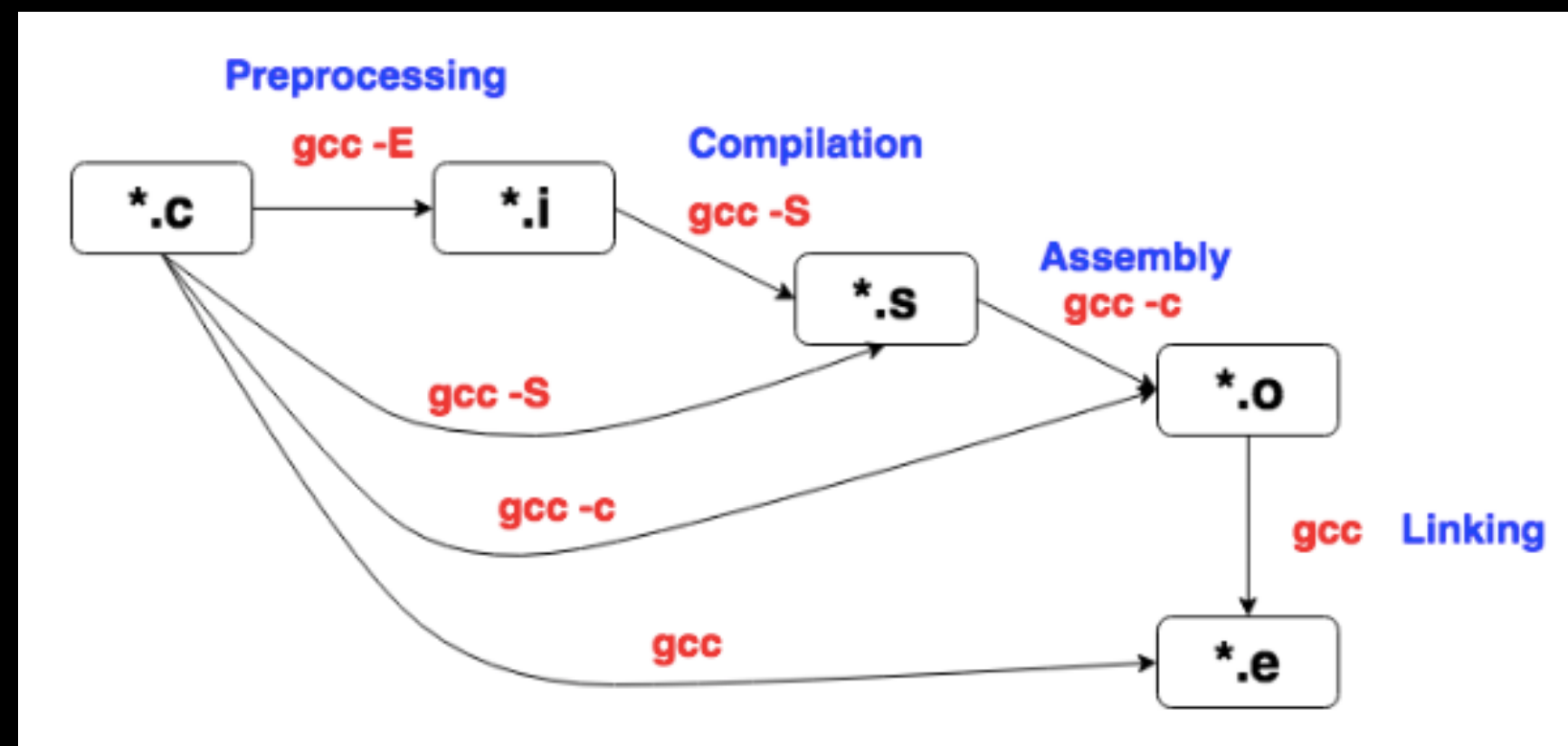
REPASO

COMPILANDO CON gcc / g++

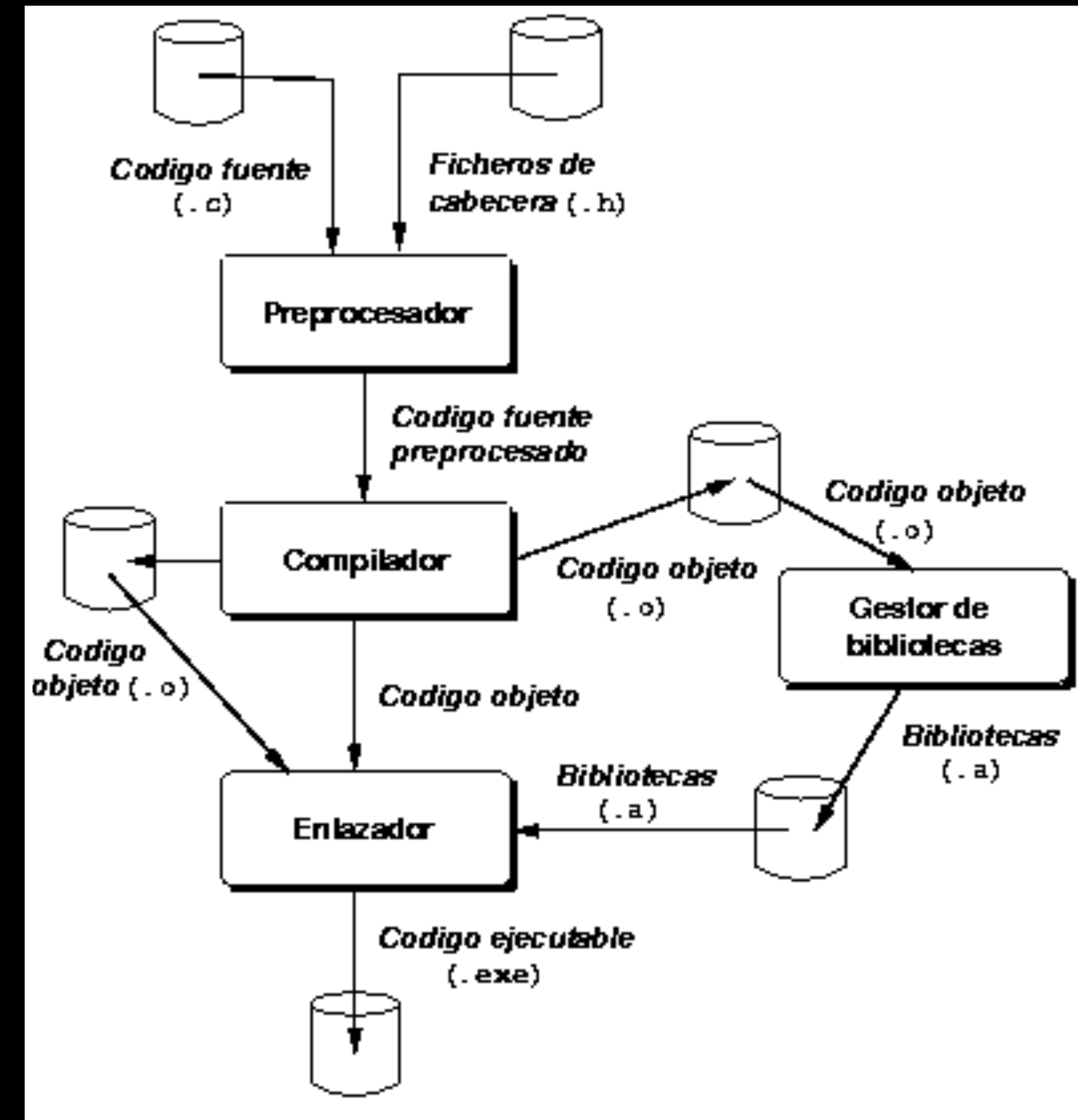
gcc / g++

Compilando un programa en C

- Para compilar los ejercicios del TP podemos utilizar gcc o g++
- Algunas operaciones



- gcc fuente.c
- gcc -o binario fuente.c
- gcc -c fuente.c
- gcc -o binario fuente.o



GESTIÓN DE DEPENDENCIAS

GESTIÓN DE DEPENDENCIAS

MAKE

EJEMPLO

MAKE FILE

Fuentes Prog

The screenshot displays the Visual Studio Code interface with three open files: `makefile`, `main.c`, and `saludo.c`. The `makefile` file is the primary focus, showing a series of build rules. A red box highlights the first three lines of the `makefile`, which are comments explaining the file's purpose and usage. The `main.c` file shows a simple C program that includes `saludo.h` and calls `saludar()`. The `saludo.c` file shows the implementation of the `saludar()` function, which prints "Hola sisop\n".

```
1  #definicion de una regla Comentario
2  #objetivo: dependencias
3  #instrucciones
4  COMPI := gcc
5  NAME_PROG := "Hola Sisop"
6  .PHONY: clean install
7  all: prod
8  prod: ./src/main.c saludo.o
9      echo "creando programa binario => binario"
10     $(COMPI) -o $(NAME_PROG) src/main.c saludo.o
11     echo "programa binario creado => binario"
12 saludo.o: src/libs/source/saludo.c src/libs/headers/saludo.h
13     echo "creando programa objeto => saludo.o"
14     $(COMPI) -c src/libs/source/saludo.c
15     echo "programa objeto creado => saludo.o"
16 clean:
17     rm -rf $(NAME_PROG) *.o
18 install:
19     mv $(NAME_PROG) $(dir)/$(NAME_PROG)
```

```
1  #include "../libs/headers/saludo.h"
2  int main()
3  {
4      saludar();
5      return 0;
6  }
```

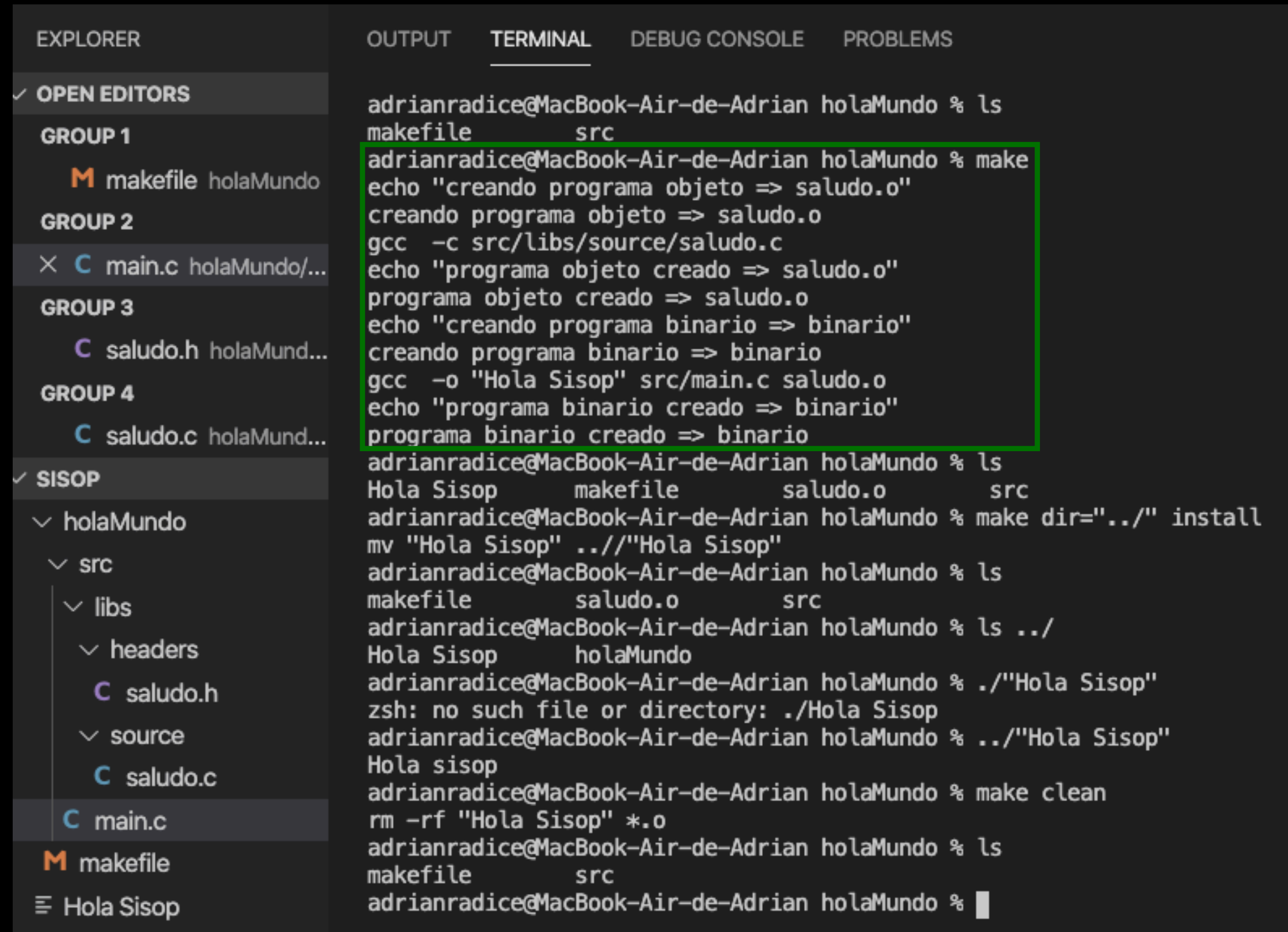
```
1  void saludar();
```

```
1  #include "../headers/saludo.h"
2  #include<stdio.h>
3  void saludar(){
4      printf("Hola sisop\n");
5  }
```

GESTIÓN DE DEPENDENCIAS

MAKE

Ejecutar makefile => \$:> make [regla]



The screenshot shows a VS Code editor with a project named 'holaMundo'. The Explorer sidebar on the left shows the file structure:

- ✓ OPEN EDITORS
 - GROUP 1
 - M makefile holaMundo
 - GROUP 2
 - × C main.c holaMundo/...
 - GROUP 3
 - C saludo.h holaMund...
 - GROUP 4
 - C saludo.c holaMund...
- ✓ SISOP
 - holaMundo
 - src
 - libs
 - headers
 - C saludo.h
 - source
 - C saludo.c
 - C main.c
 - M makefile
 - Hola Sisop

The Terminal window on the right shows the following commands and output:

```
adrianradice@MacBook-Air-de-Adrian holaMundo % ls
makefile      src
adrianradice@MacBook-Air-de-Adrian holaMundo % make
echo "creando programa objeto => saludo.o"
creando programa objeto => saludo.o
gcc -c src/libs/source/saludo.c
echo "programa objeto creado => saludo.o"
programa objeto creado => saludo.o
echo "creando programa binario => binario"
creando programa binario => binario
gcc -o "Hola Sisop" src/main.c saludo.o
echo "programa binario creado => binario"
programa binario creado => binario
adrianradice@MacBook-Air-de-Adrian holaMundo % ls
Hola Sisop    makefile      saludo.o      src
adrianradice@MacBook-Air-de-Adrian holaMundo % make dir="../" install
mv "Hola Sisop" ../"Hola Sisop"
adrianradice@MacBook-Air-de-Adrian holaMundo % ls
makefile      saludo.o      src
adrianradice@MacBook-Air-de-Adrian holaMundo % ls ../
Hola Sisop    holaMundo
adrianradice@MacBook-Air-de-Adrian holaMundo % ./"Hola Sisop"
zsh: no such file or directory: ./Hola Sisop
adrianradice@MacBook-Air-de-Adrian holaMundo % ../"Hola Sisop"
Hola sisop
adrianradice@MacBook-Air-de-Adrian holaMundo % make clean
rm -rf "Hola Sisop" *.o
adrianradice@MacBook-Air-de-Adrian holaMundo % ls
makefile      src
adrianradice@MacBook-Air-de-Adrian holaMundo %
```

Multi-Procesos

PROCESOS PESADOS

PROCESOS PESADOS

FORK

- Comportamiento = Duplica PCB
- NO COMPARTEN MEMORIA LOS PADRES CON LOS HIJOS
- fork esta incluido en unistd.h
- Retorna un pid_t
 - -1 error
 - 0 para indicar el hijo
 - >0 (pid del nuevo proceso) en el padre

C ej.c

proPesado > C ej.c > ...

```
1  #include<stdio.h>
2  #include<sys/types.h>          //pid_t
3  #include<unistd.h>             //fork
4  #include<sys/wait.h>           //wait
5  #include<stdlib.h>
6  #define ERROR_FORK -1
7  int main() {
8      pid_t pid_hijo, pid;
9      pid_hijo = fork(); //syscall para crear nuevo proceso (COPIA PCB)
10     // LOS PID SON POSITIVOS
11     if ( pid_hijo == ERROR_FORK ) {
12         perror("error al crear el nuevo proceso");
13         return 1;
14     }
15     pid = getpid();
16     if ( pid_hijo == 0 ){
17         pid_t parent_pid = getppid();
18         printf("soy %d hijo de %d\n", pid, parent_pid);
19     }
20     else {
21         printf("soy %d padre de %d\n", pid, pid_hijo);
22         wait(NULL); //join
23     }
24     return EXIT_SUCCESS;
25 }
26
```

OUTPUT

TERMINAL

DEBUG CONSOLE

PROBLEMS

```
adrianradice@MacBook-Air-de-Adrian procesoLivianos % gcc -o pesado ej.c
adrianradice@MacBook-Air-de-Adrian procesoLivianos % ./pesado
soy 26135 padre de 26136
soy 26136 hijo de 26135
adrianradice@MacBook-Air-de-Adrian procesoLivianos %
```


PROCESOS PESADOS

WAIT

- `pid_t wait (int *status)`
 - Suspende llamador hasta recibir SIGCHLD de alguno de sus hijos. Permite recuperar el estado de finalización del mismo. El retorno es el pid del hijo finalizado o -1 en caso de error. Si nos interesa info de OS, tenemos `wait3` que en su tercer parámetro almacena dicha info.
`pid_t wait3 (int *status, int options, struct rusage *rusage)`
- `pid_t waitpid (pid_t pid, int *status, int options)`
 - Igual a `wait` pero permite indicar que hijo esperar, y mediante la opción `WNOHANG` la espera es no bloquearte. Si nos interesa info de OS, tenemos `wait3` que en su tercer parámetro almacena dicha info.
`pid_t wait4 (pid_t pid, int *status, int options, struct rusage *rusage)`
- `wait` es equivalente a `waited (-1, status, 0)`

PROCESOS PESADOS

EXEC vs SYSTEM

- EXEC reemplaza la imagen del proceso llamador por el nuevo a ejecutar.
- SYSTEM es equivalente a ejecutar exec en el bloque de código (pid == 0) de un hijo creado con fork

The screenshot shows a code editor with two files: `exec.c` and `system.c`. Below the code, there are two terminal windows showing the execution of these programs.

```
proPesado > exec > C exec.c > ...
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<stdlib.h>
4  int main() {
5      char *binaryPath = "/bin/ls";
6      char *args[] = {binaryPath, NULL};
7      execv(binaryPath, args);
8      printf("ESTA LINEA YA NO SE EJECUTA\n");
9      return EXIT_SUCCESS;
10 }
11
```

```
proPesado > system > C system.c > ...
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<stdlib.h>
4  int main() {
5      char *binaryPath = "/bin/ls";
6      system(binaryPath);
7      printf("ESTA LINEA YA SE EJECUTA\n");
8      return EXIT_SUCCESS;
9  }
10
```

Terminal Output for `exec.c`:

```
adrianradice@MacBook-Pro exec % ./exec
exec    exec.c
adrianradice@MacBook-Pro exec %
```

Terminal Output for `system.c`:

```
adrianradice@MacBook-Pro system % ./system
system    system.c
ESTA LINEA YA SE EJECUTA
adrianradice@MacBook-Pro system %
```

PROCESOS LIVIANOS

PROCESOS LIVIANOS

THREAD

- COMPARTEN MEMORIA
- Necesitamos de unistd.h
- Retorna un pthread_t
- OPERACIONES
 - CREAR HILOS
 - `int pthread_create (pthread_t tid, pthread_attr_t *atributosThread, void *(*codeThread), void *ParamsThread)`
 - Si el hilo es dependiente necesitamos ejecutar el join para que se libere el bloqueo
 - `Int pthread_join (pthread_t did, void ** retornoThread)`
 - Toda función de un hilo termina con
 - `pthread_exit(void *retorno)`
 - Envio de señales con
 - `pthread_kill (pthread_t x, int signal_number)`
 - Para obtener tu propio TID
 - `pthread_t pthread_self (void)`
 - Espera no bloqueante de un semáforo
 - `pthread_mutex_trylock(phtread_mutex_t *m)`

```
C hilos.c ×
Procesos Livianos > C hilos.c > main()
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <pthread.h>
4  int global_var = 0;
5  pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER; // 1
6  void *codeThread ( void *name ) {
7      pthread_mutex_lock ( &mutex ); //P(mutex)
8      printf ("Nombre del hilo: %s \n", ( char * ) name);
9      global_var ++;
10     pthread_mutex_unlock ( &mutex );
11     pthread_exit ( 0 );
12 }
13 int main () {
14     pthread_t hilo1, hilo2; //unsigned long int
15
16     printf( "global_var= %d \n", global_var );
17
18     pthread_create ( &hilo1, NULL, &codeThread, ( void * )"HIL01");
19     pthread_create ( &hilo1, NULL, &codeThread, ( void * )"HIL02");
20
21     pthread_join( hilo1, NULL );
22     pthread_join( hilo2, NULL );
23
24     pthread_mutex_destroy ( &mutex );
25
26     printf( "global_var= %d \n", global_var );
27
28     return EXIT_SUCCESS;
29 }
```

```
OUTPUT  TERMINAL  DEBUG CONSOLE  PROBLEMS
adrianradice@MacBook-Pro Procesos Livianos % make
gcc -o "hilos" hilos.c -lpthread
adrianradice@MacBook-Pro Procesos Livianos % ./hilos
global_var= 0
Nombre del hilo: HIL02
Nombre del hilo: HIL01
global_var= 2
adrianradice@MacBook-Pro Procesos Livianos % ./hilos
global_var= 0
Nombre del hilo: HIL02
Nombre del hilo: HIL01
global_var= 2
adrianradice@MacBook-Pro Procesos Livianos % ./hilos
global_var= 0
Nombre del hilo: HIL01
Nombre del hilo: HIL02
global_var= 2
adrianradice@MacBook-Pro Procesos Livianos %
```

RC

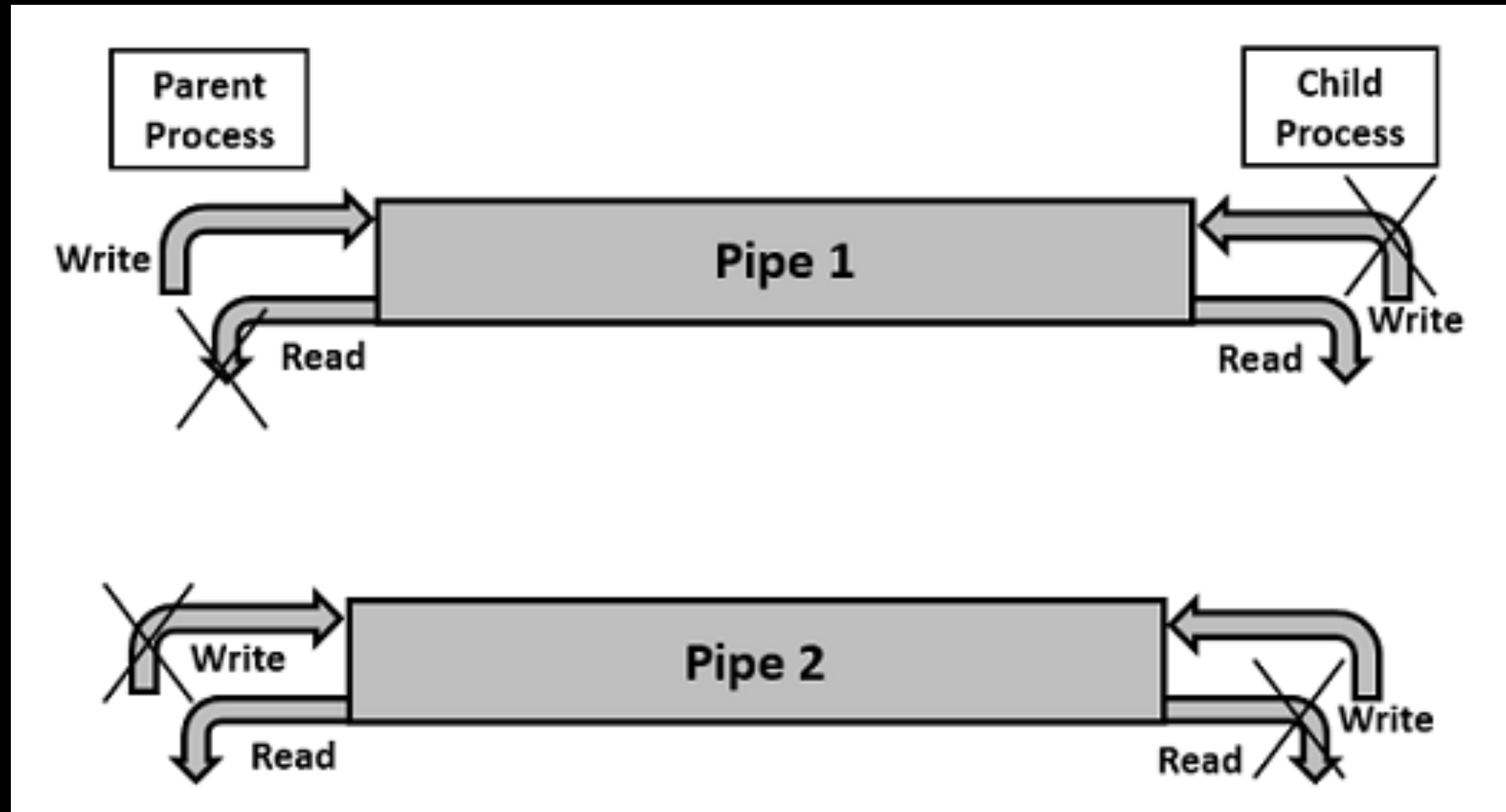
Hirschfeldt D y Radice A

COMUNICACIÓN

PIPE

PIPE

TUBERIA SIN NOMBRE



Los pipe son unidireccionales

Se usa para procesos emparentados

```
C pipe.c × M makefile
Comunicacion > PIPE > C pipe.c > main()
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<stdlib.h>
4  void validateOpenPipe(int status){
5      if (status == -1) {
6          printf("Error al crear el pipe\n");
7          exit(EXIT_FAILURE);
8      }
9  }
10 int main() {
11     int pipefds1[2], pipefds2[2];
12     int pid;
13     char pipe1writemessage[20] = "Hola hijo";
14     char pipe2writemessage[20] = "Hola PA";
15     char readmessage[20];
16     validateOpenPipe( pipe(pipefds1) );
17     validateOpenPipe( pipe(pipefds2) );
18     pid = fork();
19     if(pid == -1)
20         return -1;
21     if (pid == 0){
22         close(pipefds1[0]);
23         close(pipefds2[1]);
24         printf("PAPI DICE: %s\n", pipe1writemessage);
25         write(pipefds1[1], pipe1writemessage, sizeof(pipe1writemessage));
26         read(pipefds2[0], readmessage, sizeof(readmessage));
27         printf("PAPI ESCUCHA: %s\n", readmessage);
28     } else {
29         close(pipefds1[1]);
30         close(pipefds2[0]);
31         read(pipefds1[0], readmessage, sizeof(readmessage));
32         printf("HIJO ESCUCHA: %s\n", readmessage);
33         printf("HIJO DICE %s\n", pipe2writemessage);
34         write(pipefds2[1], pipe2writemessage, sizeof(pipe2writemessage));
35     }
36     return 0;
37 }
```

OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS

```
adrianradice@MacBook-Pro PIPE % make
gcc -o pipe pipe.c
adrianradice@MacBook-Pro PIPE % ./pipe
PAPI DICE: Hola hijo
HIJO ESCUCHA: Hola hijo
HIJO DICE Hola PA
PAPI ESCUCHA: Hola PA
adrianradice@MacBook-Pro PIPE %
```

FIFO

TUBERIA CON NOMBRE

Dado que son archivos con nombre a Dif de los pipe, es que pueden usarse para procesos no emparentados.

Admite comunicación bidireccional pero debe sincronizarse el uso del canal

```
C servidor.c x
Comunicacion > FIFO > C servidor.c > main()
1  #include <stdio.h>
2  #include <sys/stat.h>
3  #include <sys/types.h>
4  #include <fcntl.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include "./config.h"
8  int main() {
9      char readbuf[TAMBUF];
10     int read_bytes;
11     //crear fifo en caso de que no exista
12     mkfifo(FIFO_FILE, S_IFIFO|0640);
13     //abrir el fifo
14     int fd = open(FIFO_FILE, O_RDWR);
15     while(1) {
16         //leer fifo
17         read_bytes = read(fd, readbuf, sizeof(readbuf));
18         //poner fin de cadena para no imprimir basura
19         readbuf[read_bytes] = '\0';
20         printf("FIFOSEVER: RECV: %s\n", readbuf);
21         //si el cliente envia end se termina la com
22         if (strcmp(readbuf, "end") == 0)
23             break;
24         printf("FIFOSEVER: SEND %s\n", readbuf);
25         //escribir lo que recivi
26         write(fd, readbuf, strlen(readbuf));
27         //hasta ver sem, es para no recuperar lo que escribi
28         sleep(2);
29     }
30     close(fd);
31     return 0;
32 }
33

C cliente.c x
Comunicacion > FIFO > C cliente.c > ...
1  #include <stdio.h>
2  #include <sys/stat.h>
3  #include <sys/types.h>
4  #include <fcntl.h>
5  #include <unistd.h>
6  #include <string.h>
7  #include "./config.h"
8  int main() {
9      int stringlen;
10     int read_bytes;
11     char readbuf[TAMBUF];
12     int fd = open(FIFO_FILE, O_CREAT|O_RDWR);
13     while (1) {
14         printf("EscribaMensaje: ");
15         fgets(readbuf, sizeof(readbuf), stdin);
16         stringlen = strlen(readbuf);
17         readbuf[stringlen - 1] = '\0';
18         if ( strcmp(readbuf, "end") == 0)
19             break;
20         write(fd, readbuf, strlen(readbuf));
21         printf("FIFOCLIENT: Send %s\n", readbuf);
22         //hasta ver sem, es para no recuperar lo que escribi
23         sleep(1);
24         read_bytes = read(fd, readbuf, sizeof(readbuf));
25         readbuf[read_bytes] = '\0';
26         printf("FIFOCLIENT: RECV: %s (%d chars)\n", readbuf,
27             (int)strlen(readbuf));
28     }
29     write(fd, readbuf, strlen(readbuf));
30     printf("FIFOCLIENT: SEND: %s\n", readbuf);
31     close(fd);
32     return 0;
33 }
34
OUTPUT TERMINAL DEBUG CONSOLE PROBLEMS
1: zsh, zsh
adrianradice@MacBook-Pro FIFO % ./servidor
FIFOSEVER: RECV: hola
FIFOSEVER: SEND hola
FIFOSEVER: RECV: como estas
FIFOSEVER: SEND como estas
FIFOSEVER: RECV: end
adrianradice@MacBook-Pro FIFO %

adrianradice@MacBook-Pro FIFO % ./cliente
EscribaMensaje: hola
FIFOCLIENT: Send hola
FIFOCLIENT: RECV: hola (4 chars)
EscribaMensaje: como estas
FIFOCLIENT: Send como estas
FIFOCLIENT: RECV: como estas (10 chars)
EscribaMensaje: end
FIFOCLIENT: SEND: end
adrianradice@MacBook-Pro FIFO %
```

Memoria compartida

Memoria compartida

CLIENTE - SERVIDOR

```
C memo.c C server.c X
Comunicacion > Memoria Compartida > C server.c > main()
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6 #include <unistd.h>
7 #define SHMSZ 27
8 main()
9 {
10     char c;
11     int shmid;
12     key_t key;
13     char *shm, *s;
14     key = 5678;
15     if ((shmid = shmget(key, SHMSZ, IPC_CREAT | 0666)) < 0) {
16         perror("shmget");
17         exit(1);
18     }
19     if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
20         perror("shmat");
21         exit(1);
22     }
23     s = shm;
24     for (c = 'a'; c <= 'z'; c++)
25         *s++ = c;
26     *s = NULL;
27     while (*shm != '*')
28         sleep(1);
29     shmdt ( shm );
30     shmctl ( shmid, IPC_RMID, NULL );
31     exit(0);
32 }
```

```
C cliente.c X M makefile
Comunicacion > Memoria Compartida > C cliente.c > main()
1 #include <sys/types.h>
2 #include <sys/ipc.h>
3 #include <sys/shm.h>
4 #include <stdio.h>
5 #include <stdlib.h>
6
7 #define SHMSZ 27
8
9 main()
10 {
11     int shmid;
12     key_t key;
13     char *shm, *s;
14     key = 5678;
15     if ((shmid = shmget(key, SHMSZ, 0666)) < 0) {
16         perror("shmget");
17         exit(1);
18     }
19     if ((shm = shmat(shmid, NULL, 0)) == (char *) -1) {
20         perror("shmat");
21         exit(1);
22     }
23     for (s = shm; *s != NULL; s++)
24         putchar(*s);
25     putchar('\n');
26     *shm = '*';
27     exit(0);
28 }
```


Semáforos

SEMAFOROS POSIX

Sincronizando procesos

```
C semA.c • makefile
Semaforos > C semA.c > main()
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<semaphore.h>
4  #include<fcntl.h>
5  int main(){
6      sem_t *x = sem_open("/x",O_CREAT|O_EXCL,0666,1);
7      sem_t *y = sem_open("/y",O_CREAT|O_EXCL,0666,0);
8      for (int i=0; i<10; i++){
9          sem_wait(x);
10         sleep(1);
11         printf("A\n");
12         sem_post(y);
13     }
14     sem_close(x);
15     sem_close(y);
16     sem_unlink("/x");
17     sem_unlink("/y");
18     return 0;
19 }
```

```
C semB.c ×
Semaforos > C semB.c > main()
1  #include<stdio.h>
2  #include<unistd.h>
3  #include<semaphore.h>
4  #include<fcntl.h>
5  int main(){
6      sem_t *x = sem_open("/x",0);
7      sem_t *y = sem_open("/y",0);
8      for (int i=0; i<10; i++){
9          sem_wait(y);
10         sleep(1);
11         printf("B\n");
12         sem_post(x);
13     }
14     sem_close(x);
15     sem_close(y);
16     return 0;
17 }
```

SOCKET

Socket

Comunicación red

```
C server.c x
Comunicacion > Socket > C server.c > main(int, char * [])
1  #include "./server.h"
2  struct sockaddr_in sa;
3  void setServer(int socket_id,int port,int max_queue)
4  {
5      bzero((char *) &sa, sizeof(struct sockaddr_in));
6      sa.sin_family = AF_INET;
7      sa.sin_port = htons(port);
8      sa.sin_addr.s_addr = INADDR_ANY;
9      bind(socket_id,(struct sockaddr *)&sa,sizeof(struct sockaddr_in));
10     listen(socket_id,max_queue);
11 };
12 int main(int argc, char *argv[])
13 {
14     int habilitar = 1;
15     socklen_t cl=sizeof(struct sockaddr_in);
16     struct sockaddr_in ca;
17     int server_socket;
18     int client_socket;
19     char buffer[TAM_BUF];
20     if((server_socket=socket(AF_INET,SOCK_STREAM,0)) ==-1)
21         return 1;
22     if (setsockopt(server_socket, SOL_SOCKET,
23         SO_REUSEADDR, &habilitar, sizeof(int)) < 0)
24         return 1;
25     setServer(server_socket,PORT,MAX_QUEUE);
26     client_socket=accept(server_socket,(struct sockaddr *) &ca, &cl);
27     send(client_socket,"BIENVENIDO",10,0);
28     bzero(buffer,TAM_BUF);
29     recv(client_socket,buffer,TAM_BUF,0);
30     printf("MSG RECV: %s\n",buffer);
31     sprintf(buffer,"chau");
32     send(client_socket, buffer, strlen(buffer), 0);
33     close(client_socket);
34     close(server_socket);
35     return 0;
36 };
37

C cliente.c x
Comunicacion > Socket > C cliente.c > main(int, char * [])
1  #include "./cliente.h"
2  struct sockaddr_in sa;
3  void set(const char *y,int z)
4  {
5      bzero(&(sa.sin_zero),8);
6      sa.sin_family = AF_INET;
7      sa.sin_port = htons(z);
8      sa.sin_addr.s_addr= inet_addr(y);
9  };
10 int main(int argc, char *argv[])
11 {
12     int x;
13     char buffer[TAM_BUF];
14     if((x=socket(AF_INET,SOCK_STREAM,0))==-1)
15         return 1;
16     set(IP,PORT);
17     if(connect(x,(struct sockaddr *) &sa,sizeof(sa))==-1)
18         return 1;
19     bzero(buffer,TAM_BUF);
20     recv(x,buffer,TAM_BUF,0);
21     printf("MSG SERVER: %s\n",buffer);
22     printf("MSG SEND:");
23     scanf("%s",buffer);
24     send(x,buffer,strlen(buffer),0);
25     bzero(buffer,TAM_BUF);
26     recv(x,buffer,TAM_BUF,0);
27     printf("%s",buffer);
28     close(x);
29     return 0;
30 }
```

GRACIAS