Programación Concurrente y Tiempo Real

Laboratorio - Práctica 1
Gestión de Procesos

D. Vallejo, M.A. Redondo, J.A. Albusac,
C. Villarrubia, C. Glez, J. Ruiz

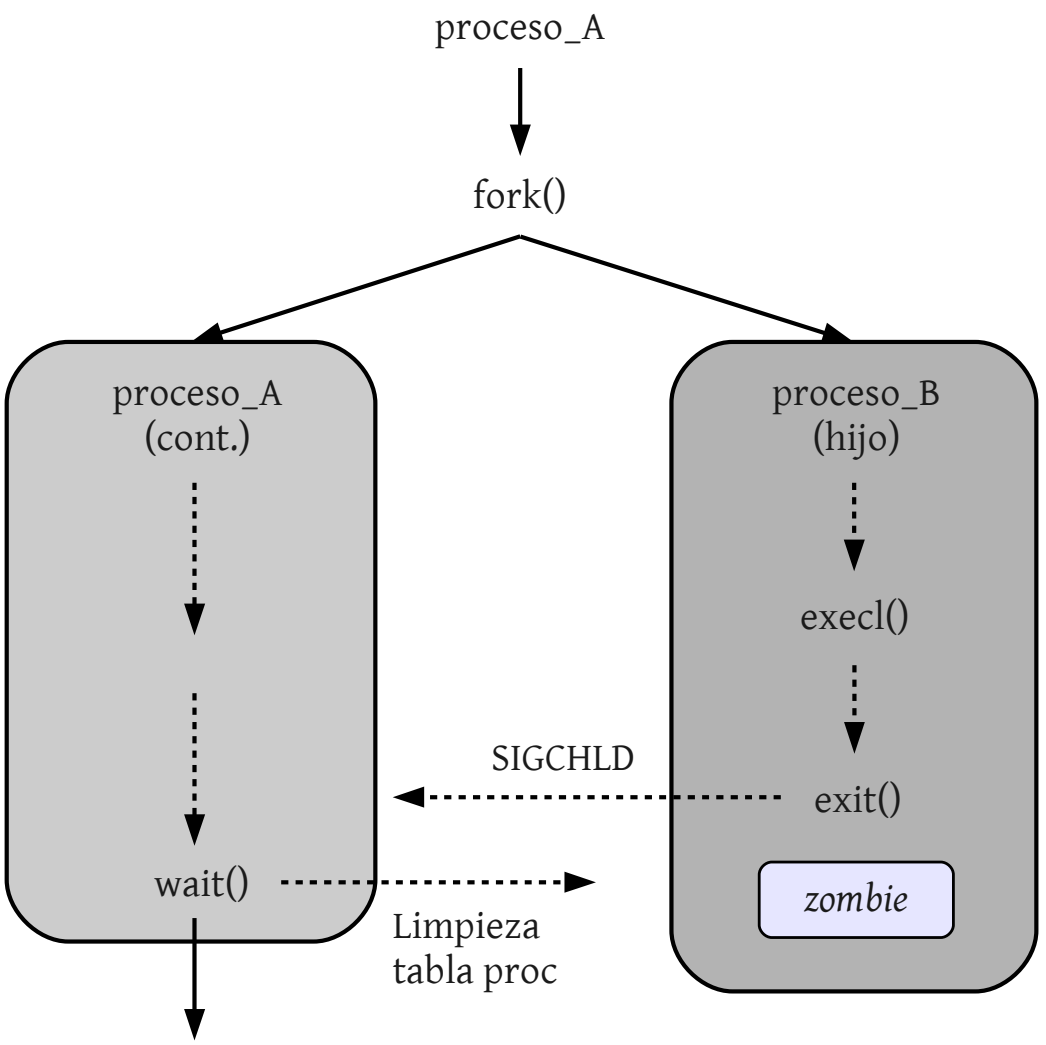
Escuela Superior de Informática

Universidad de Castilla-La Mancha

Práctica 1. Gestión de Procesos

1. Primitivas básicas POSIX
2. Modelo de prueba de laboratorio
3. Implementación
4. Bibliografía

Creación de procesos :: fork() + exec()



Creación de procesos

fork()

```
#include <sys/types.h>
#include <unistd.h>

/* 0 to the child; child's ID to the parent */
pid_t fork (void);
```

Exec family

```
#include <unistd.h>

int execl  (const char *path, const char *arg, ...);
int execlp (const char *file, const char *arg, ...);
int execle (const char *path, const char *arg, ...,
            char *const envp[]);
```

Creación de procesos :: fork() + exec()

Ejemplo. Padre crea n hijos

Parent

```
/* parent.c */

pid_t pids[N];
char sNChildren[3];
int i;

sprintf(sNChildren, "%d", N);
for (i = 0; i < N; i++)
  switch(pids[i] = fork()) {
  case 0:  /* Child's code */
    execl("./exec/child", "child", sNChildren, NULL);
  }
```

Creación de procesos :: fork() + exec()

Child

```
/* child.c */

int main(int argc, char *argv[]) {
  run(argv[1]);
  return 0;
}

void run(const char *num_brothers) {
  printf("I'm %ld and I have %d brothers!\n",
    (long)getpid(), atoi(num_brothers) - 1);
}
```

Espera y tratamiento de señales

wait primitives

```
#include <sys/types.h>
#include <sys/wait.h>

pid_t wait    (int *status);
pid_t waitpid (pid_t pid, int *status, int options);
```

wait example

```
/* Espera terminación de hijos... */
 for (i = 0; i < NUM_HIJOS; i++)
   waitpid(pids[i], 0, 0);
```

Espera y tratamiento de señales

signal/kill primitives

```
#include <signal.h>

typedef void (*sighandler_t)(int);
sighandler_t signal (int signum, sighandler_t handler);
int kill (pid_t pid, int sig);
```

Espera y tratamiento de señales

Parent

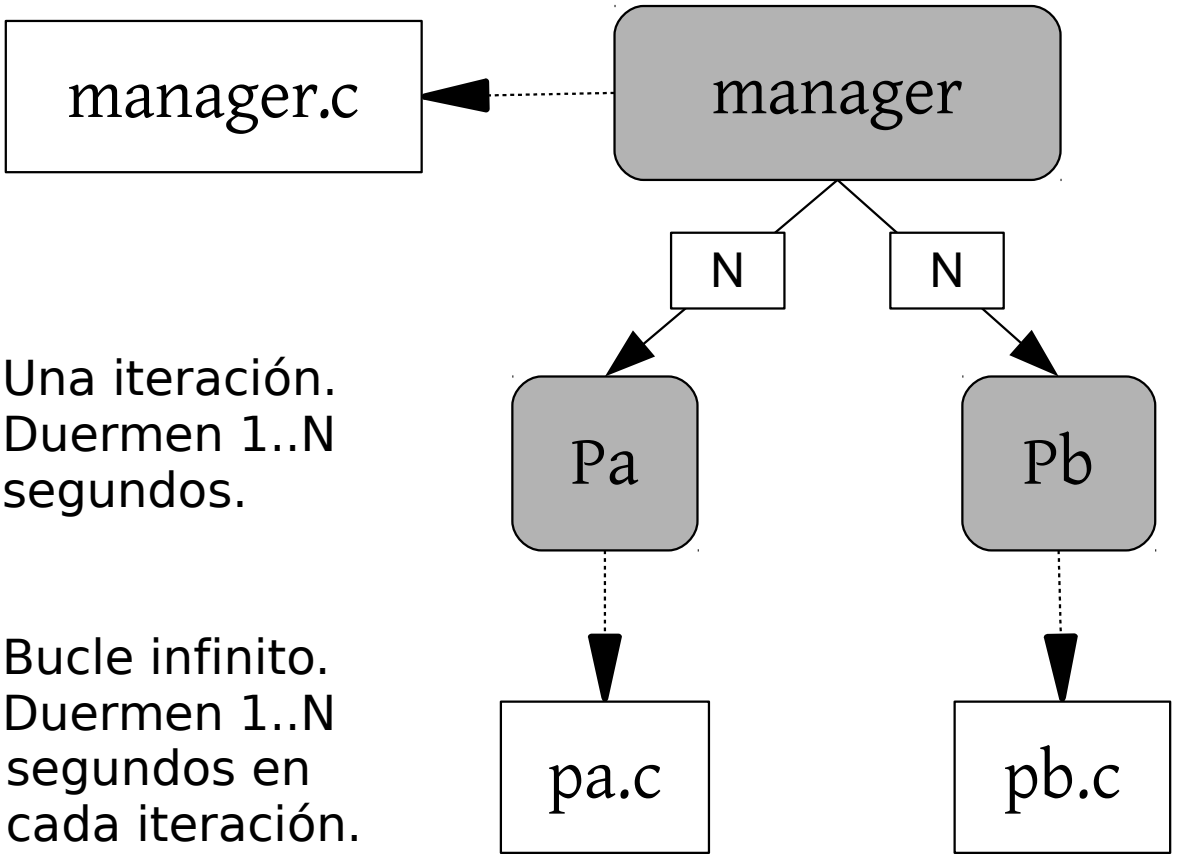
```
/* Ctrol+C management */
if (signal(SIGINT, signal_handler) == SIG_ERR) {
  fprintf(stderr, "Error installing sig hand.\n");
  exit(EXIT_FAILURE);
}

void signal_handler (int signo) {
  printf("\Program termination (Control + C)\n");
  terminate_processes();
  free_resources();
  exit(EXIT_SUCCESS);
}
```

Contexto

- Problema tipo prueba de laboratorio.
- Se suministra plantilla de código fuente.
- ~70 Minutos.
- No se permiten libros o apuntes.
- Competencias a evaluar:
  - Gestión básica de procesos.
  - Dominio de primitivas POSIX.
  - Construcción automática con Make.

Ejemplo de Uso



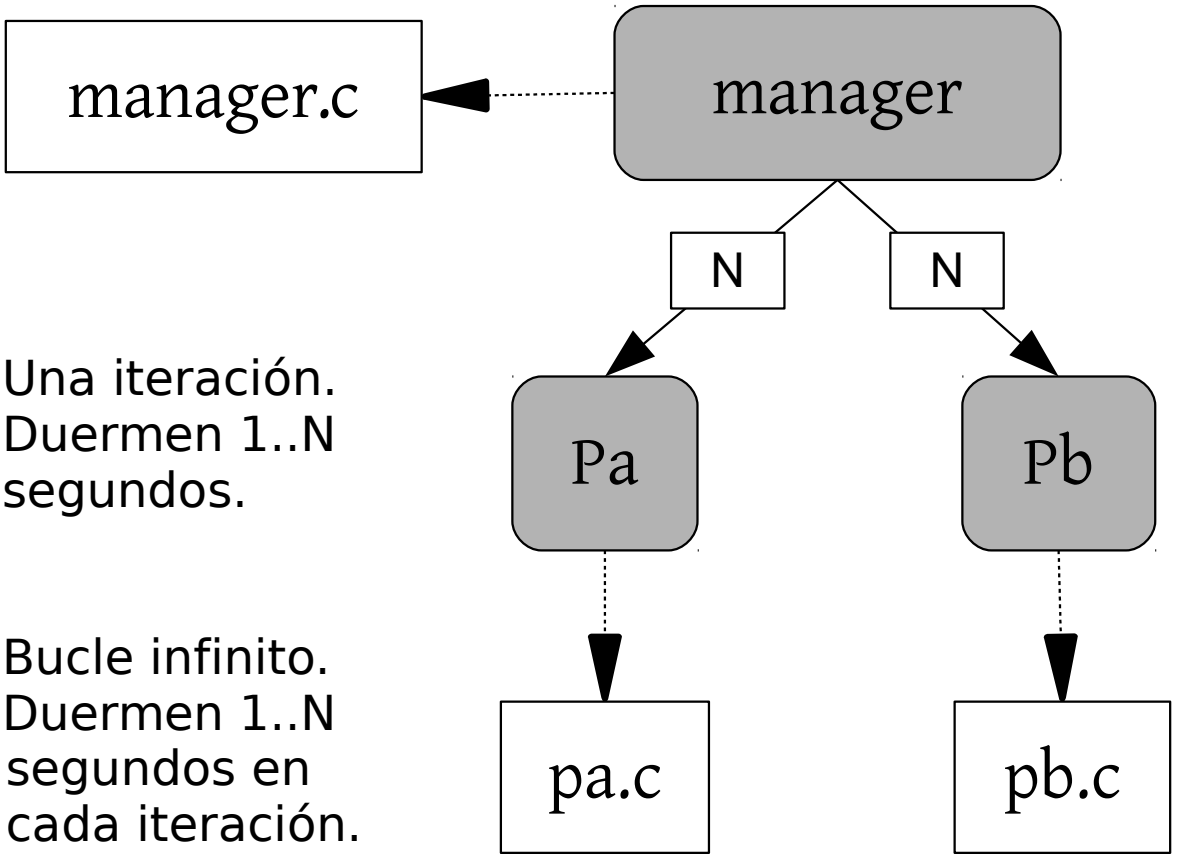
Pa

- Una iteración.
- Duermen 1..N segundos.

Pb

- Bucle infinito.
- Duermen 1..N segundos en cada iteración.

Ejemplo de Uso



Pa
- Una iteración.
- Duermen 1..N segundos.

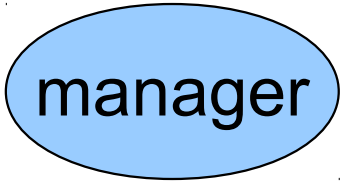
Pb
- Bucle infinito.
- Duermen 1..N segundos en cada iteración.

Parada:
- Finalizan los Pa.
- Usuario: [Ctrl+c]

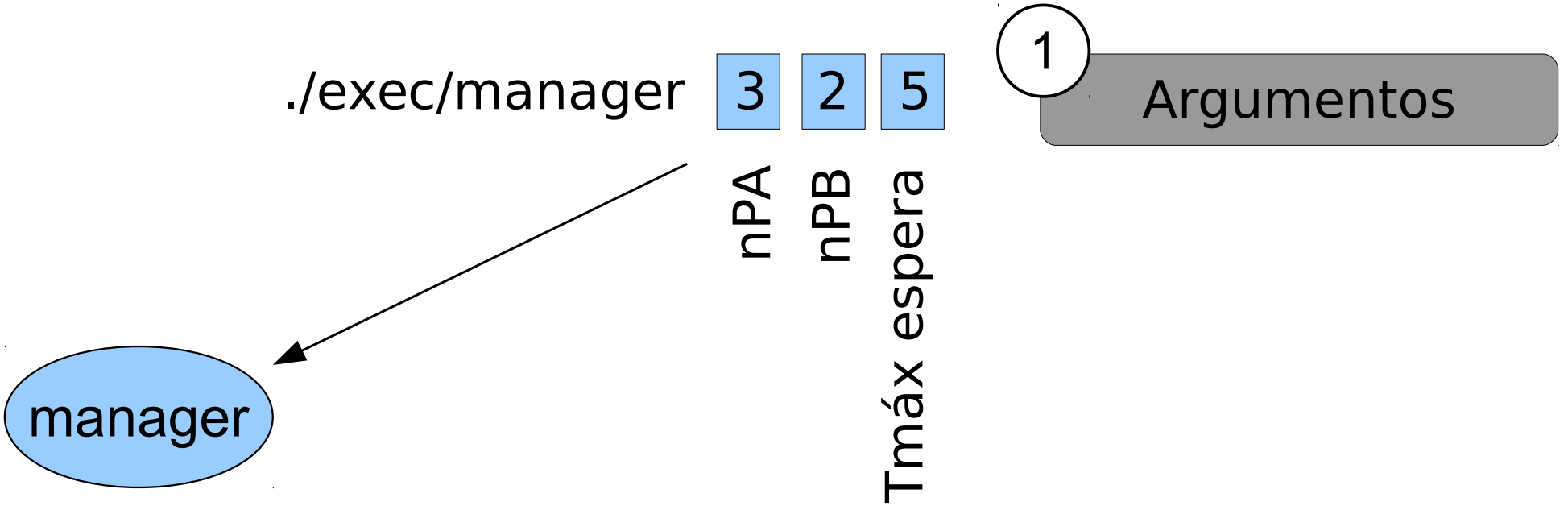
Fin correcto:
- Captura de señal.
- Propagación de señal.
- Simular liberación de recursos.
- Matar procesos.

Flujo de trabajo

./exec/manager 3 2 5

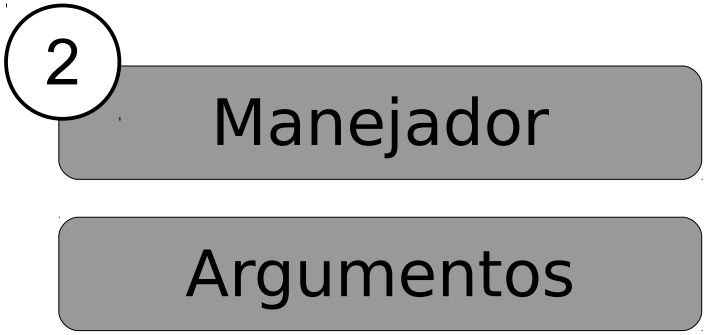


Flujo de trabajo



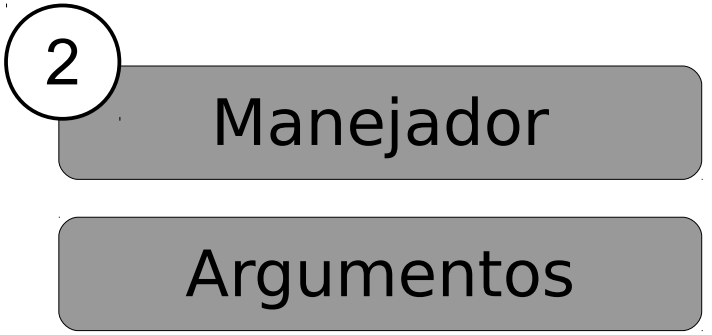
Flujo de trabajo

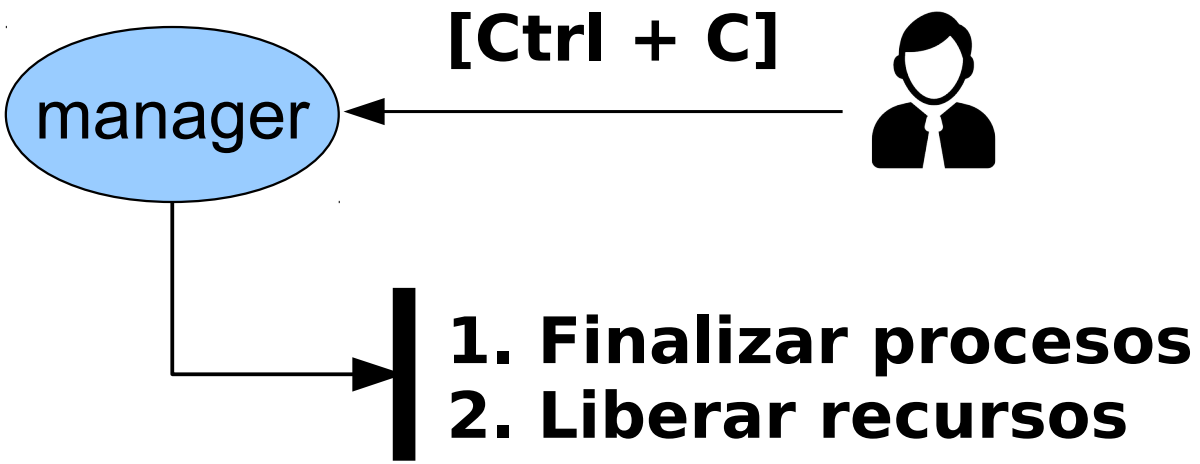
./exec/manager 3 2 5



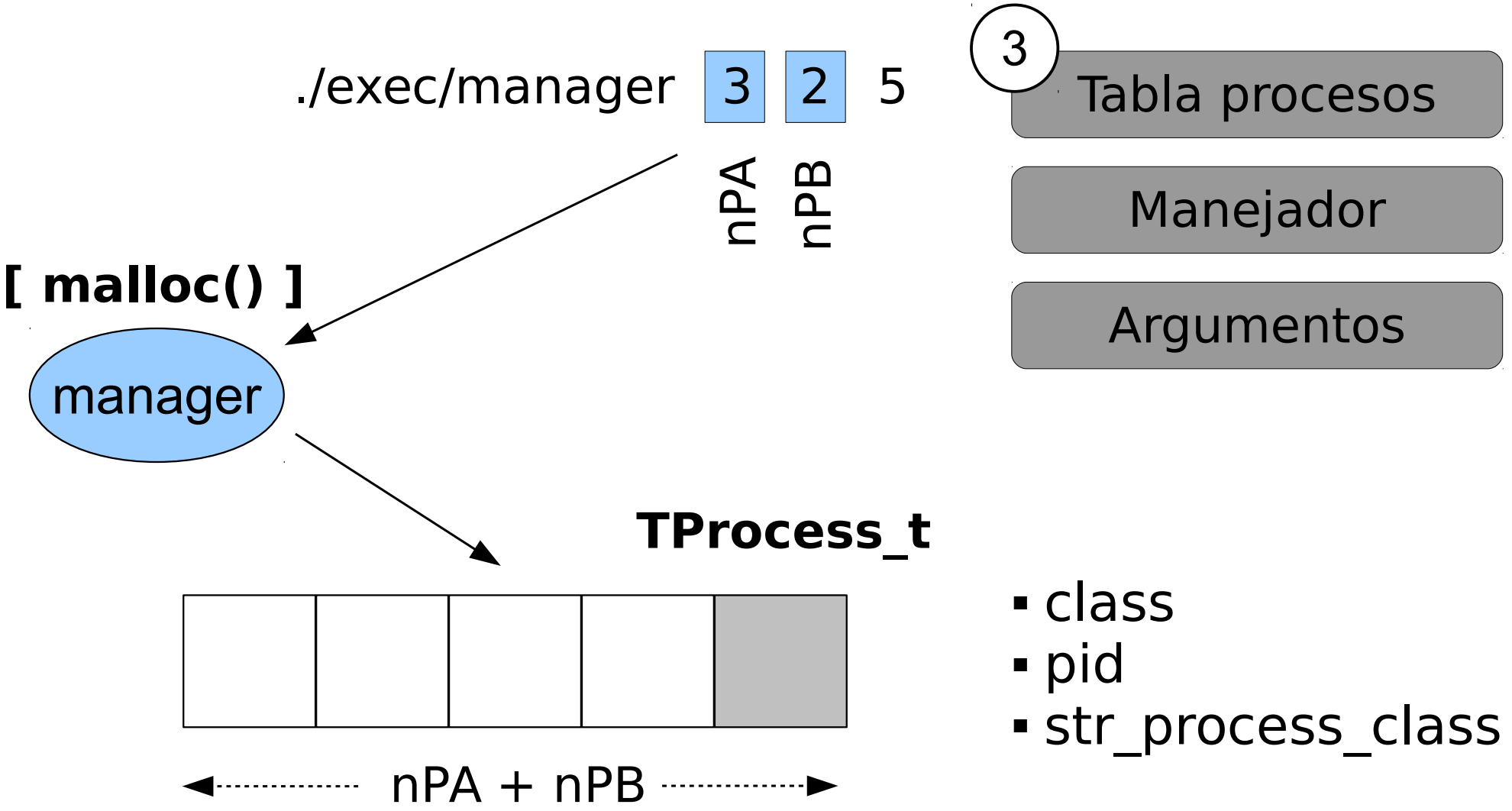
Flujo de trabajo

./exec/manager 3 2 5



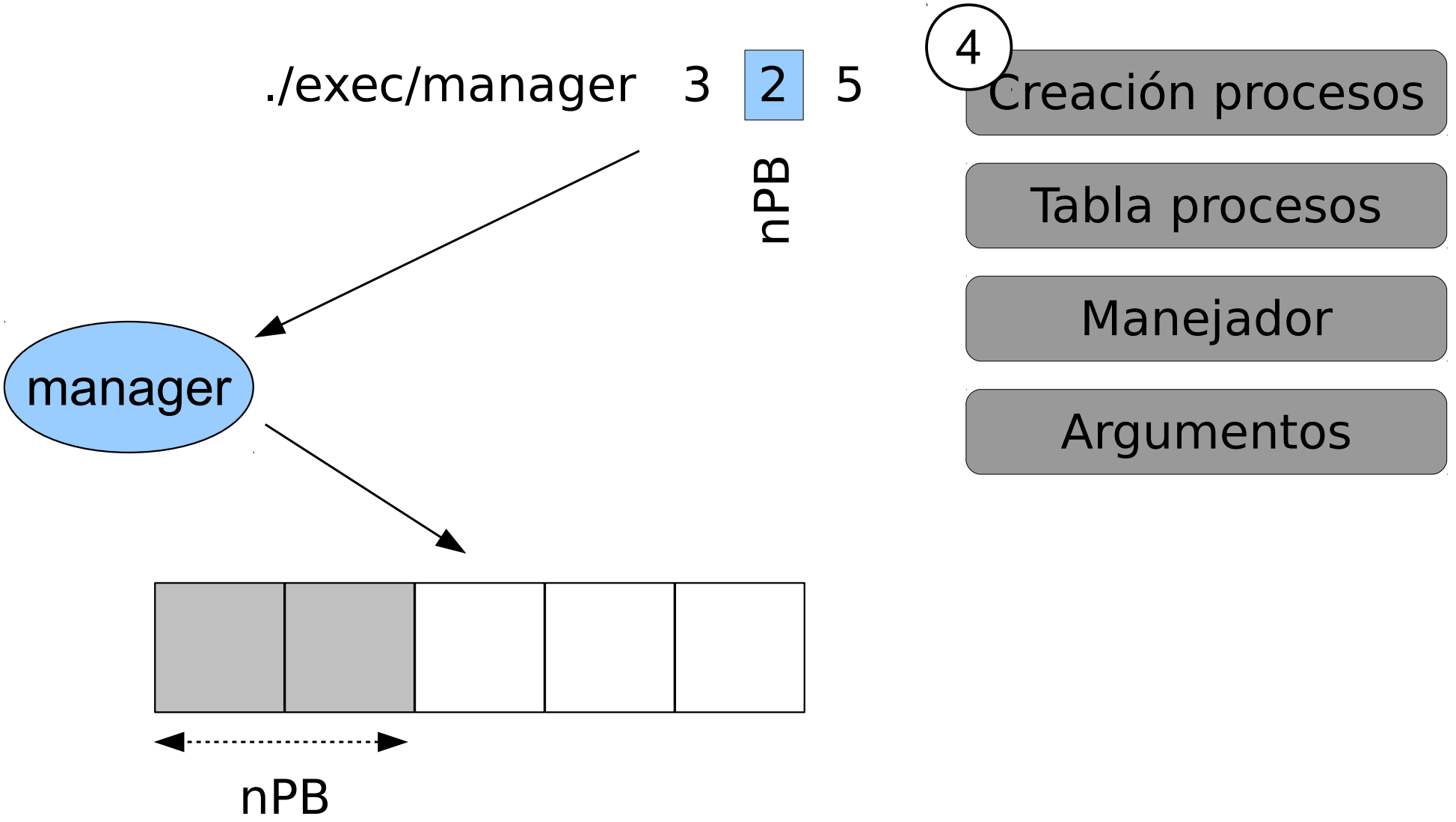


Flujo de trabajo

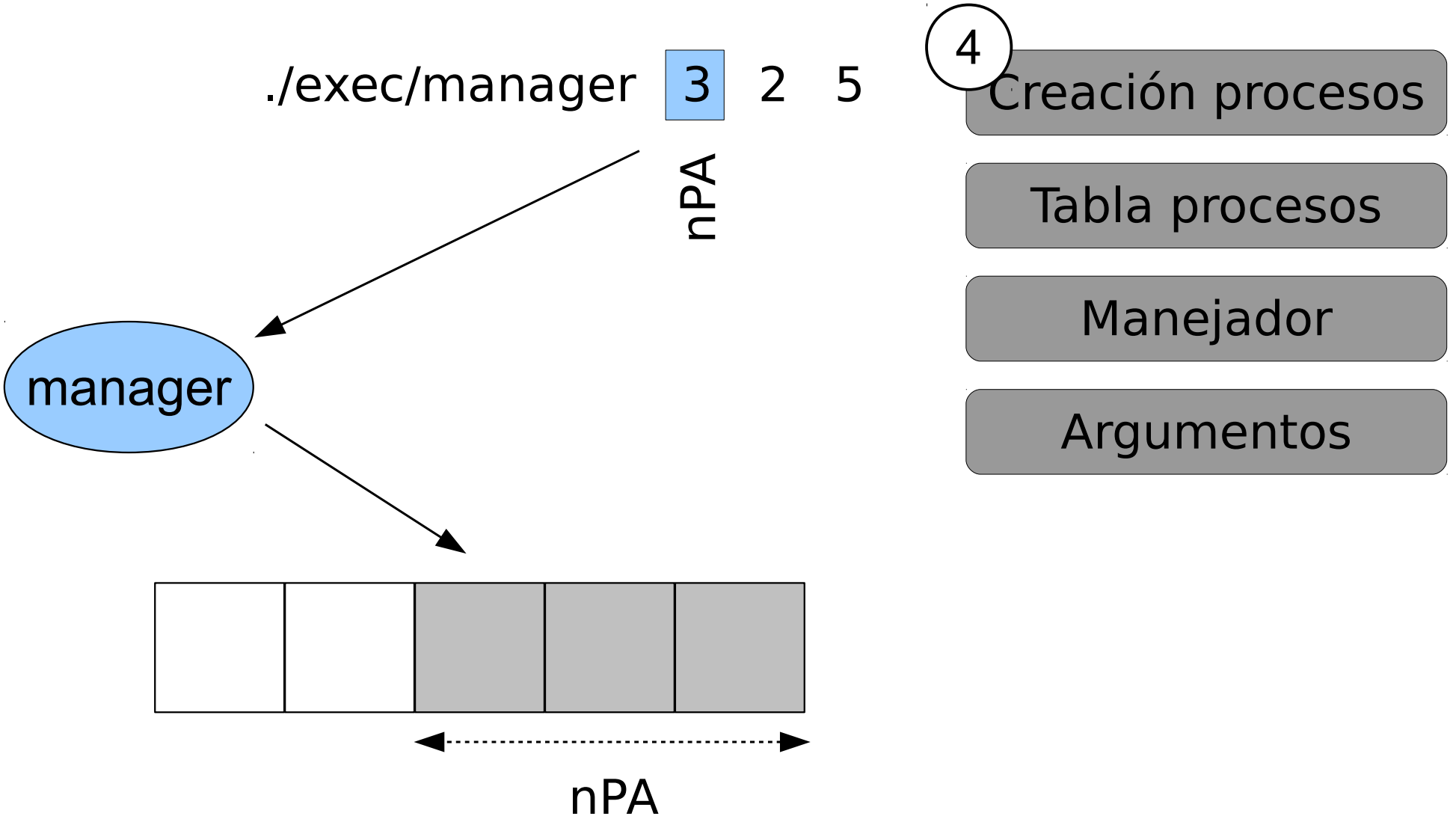


- class
- pid
- str_process_class

Flujo de trabajo



Flujo de trabajo

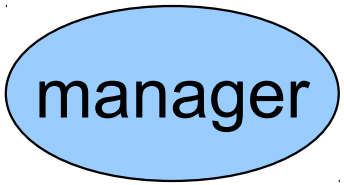


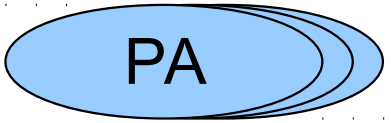
Flujo de trabajo

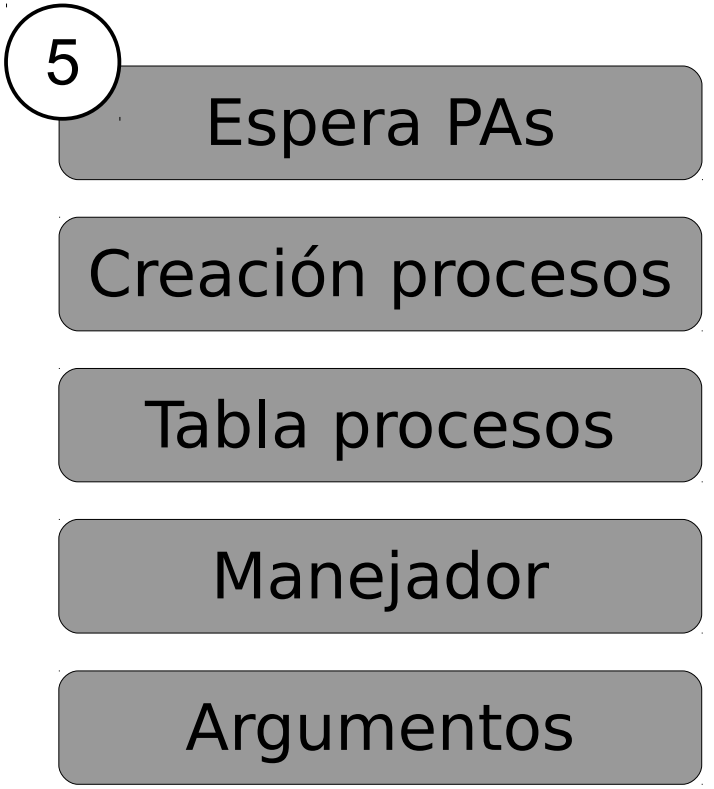
./exec/manager 3 2 5

nPA

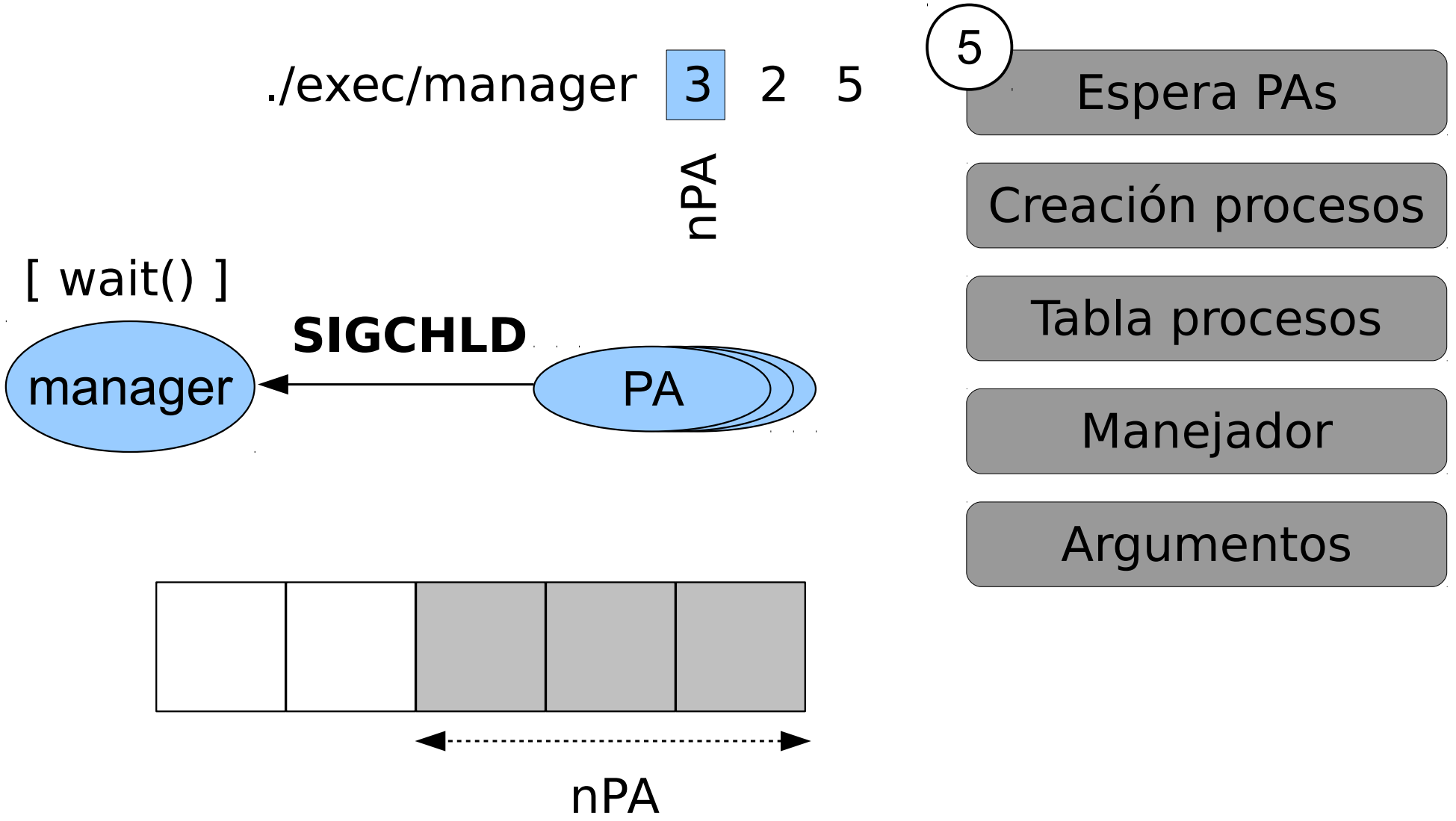
[wait()]



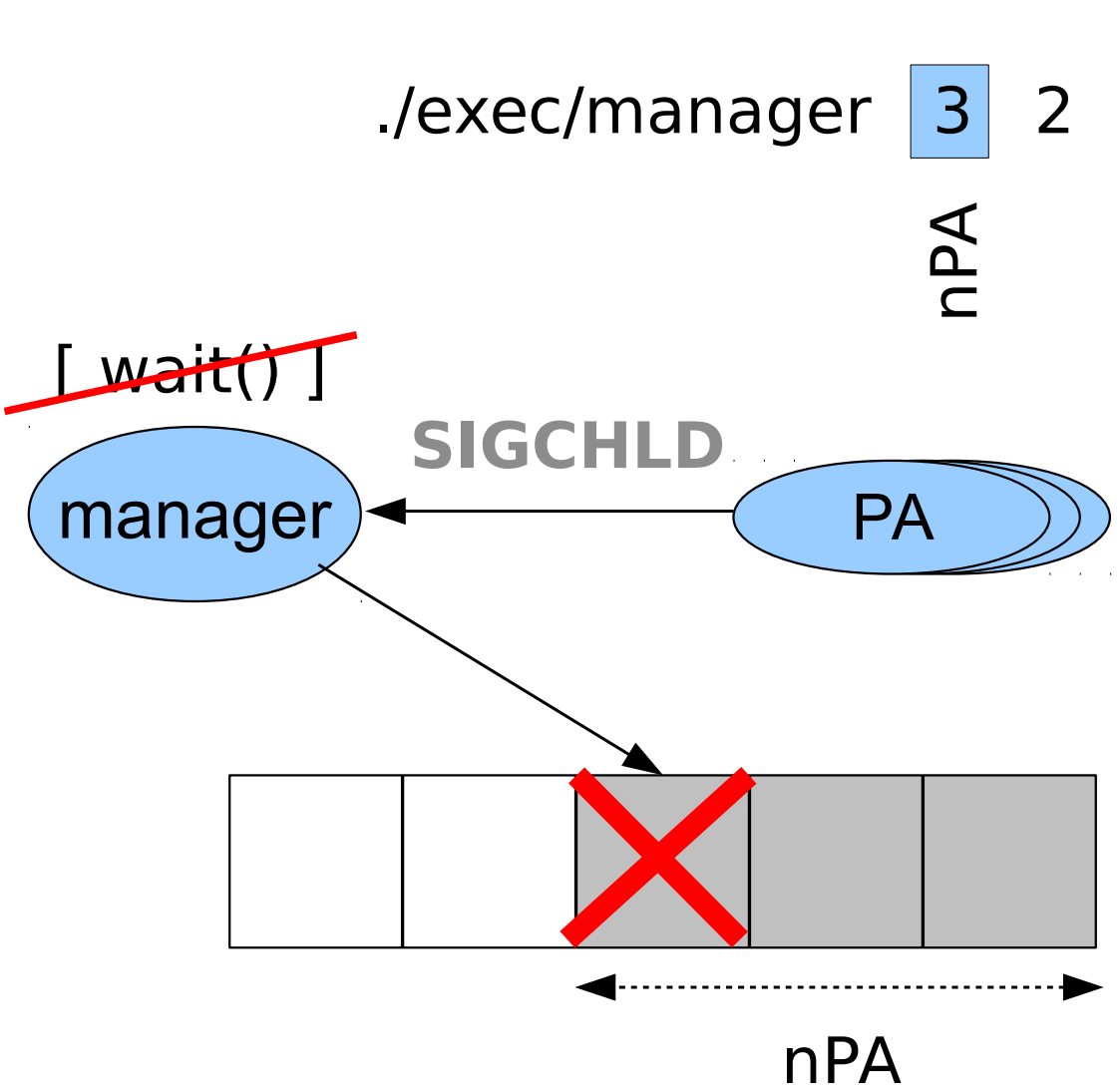




Flujo de trabajo



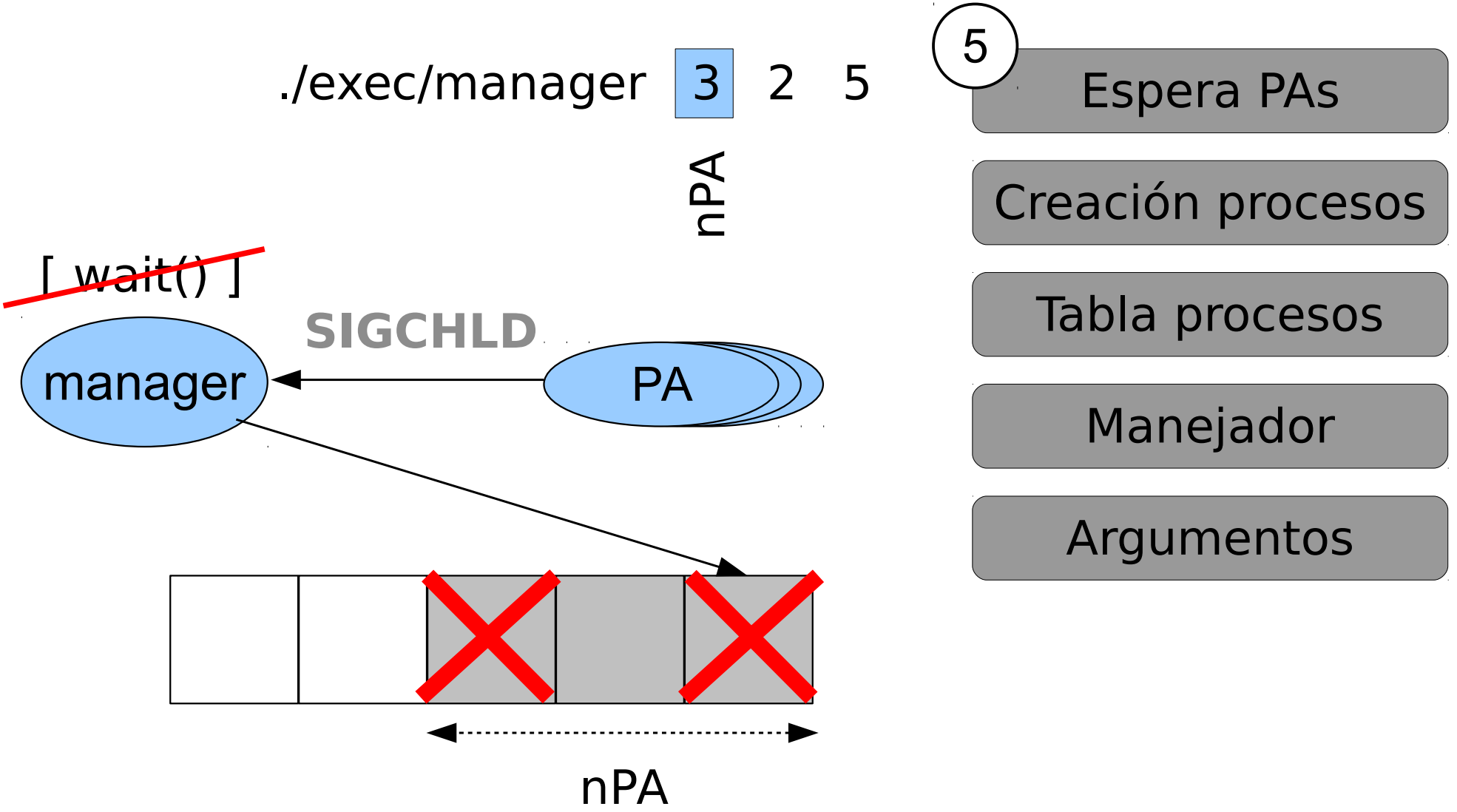
Flujo de trabajo



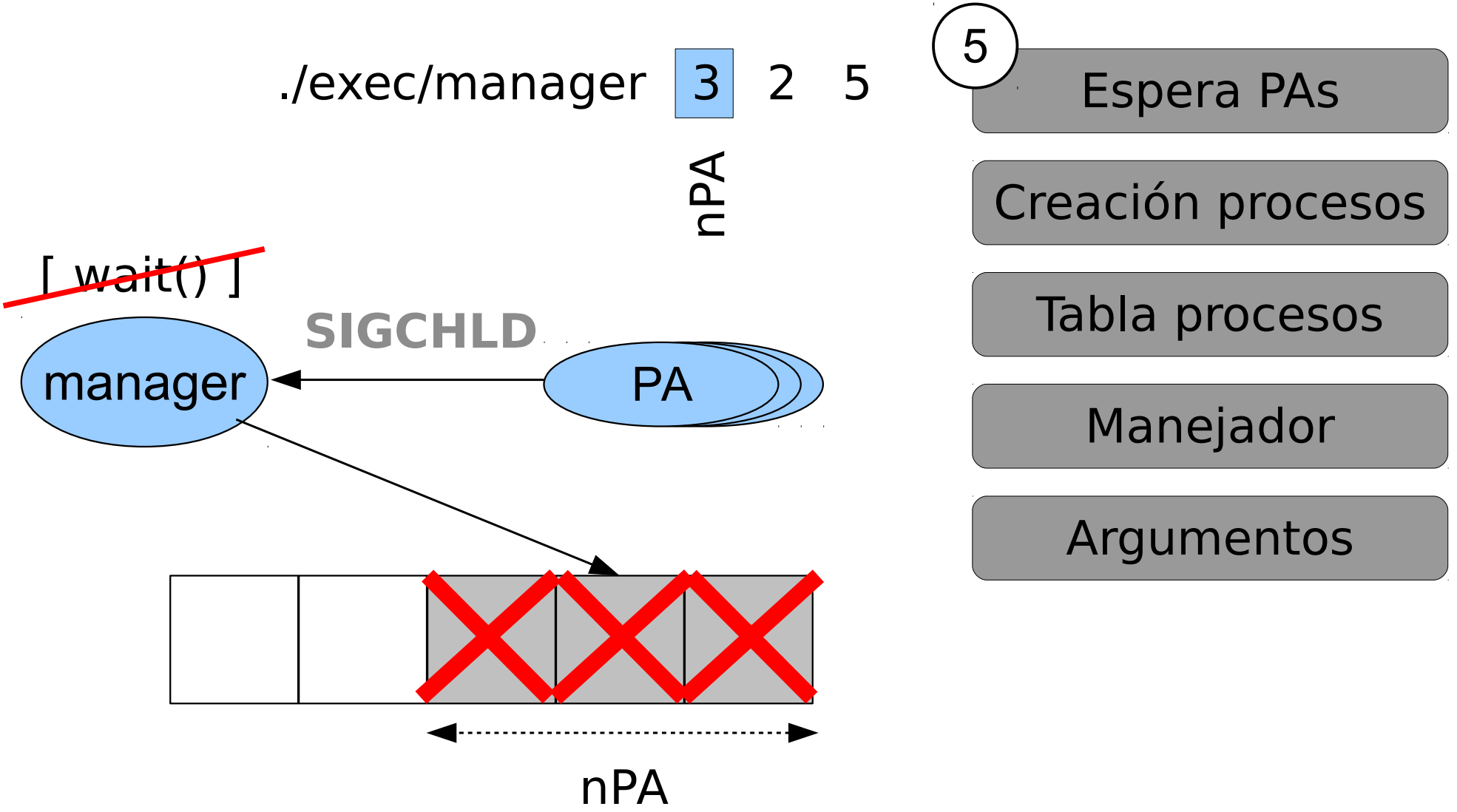
5

- Espera PAs
- Creación procesos
- Tabla procesos
- Manejador
- Argumentos

Flujo de trabajo



Flujo de trabajo



Flujo de trabajo

./exec/manager 3 2 5

nPB

6

Finalizar PBs

Espera PAs

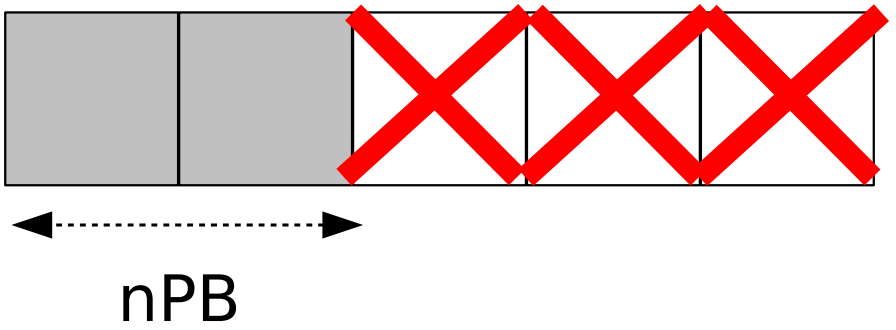
Creación procesos

Tabla procesos

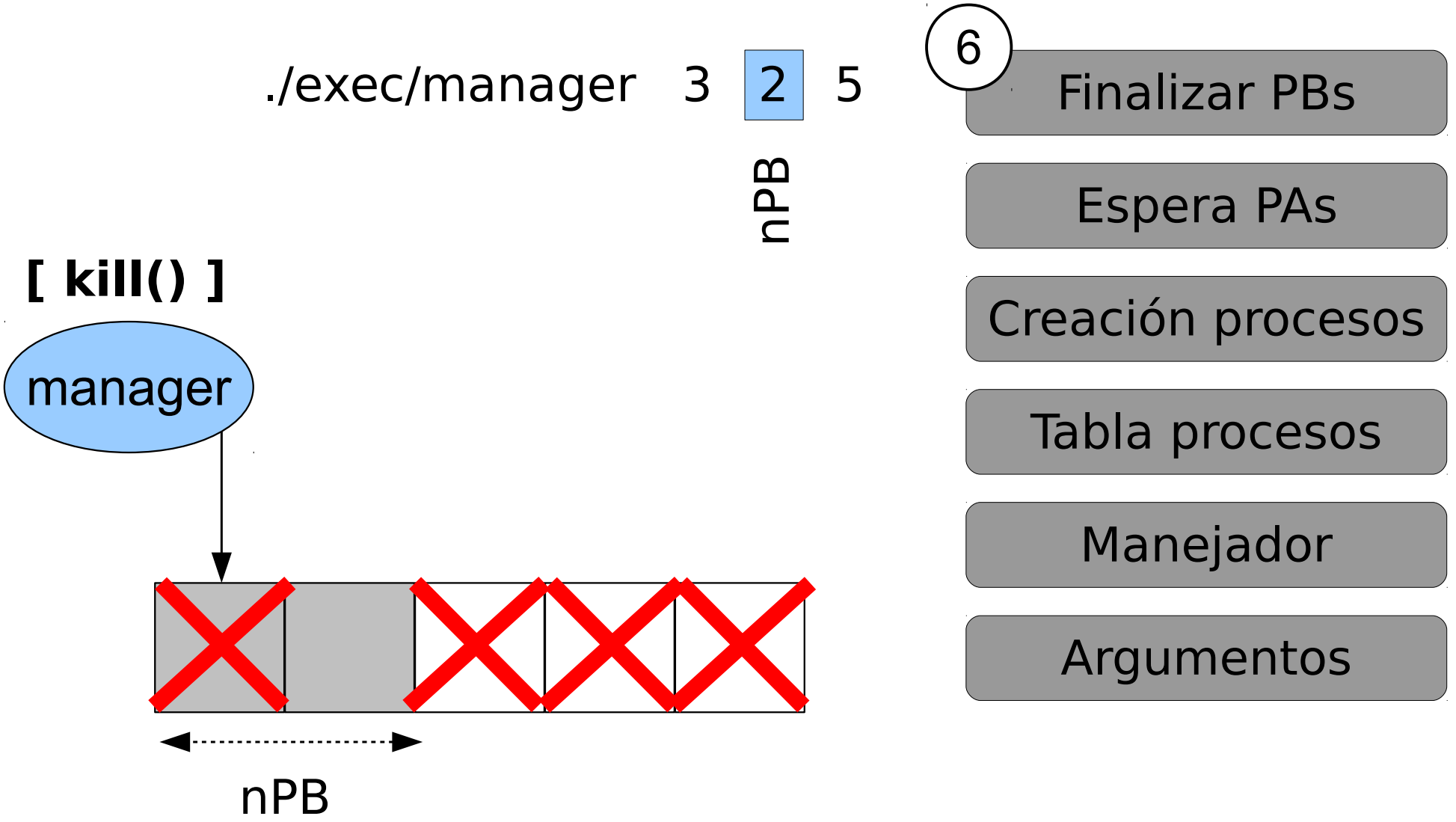
Manejador

Argumentos

manager



Flujo de trabajo

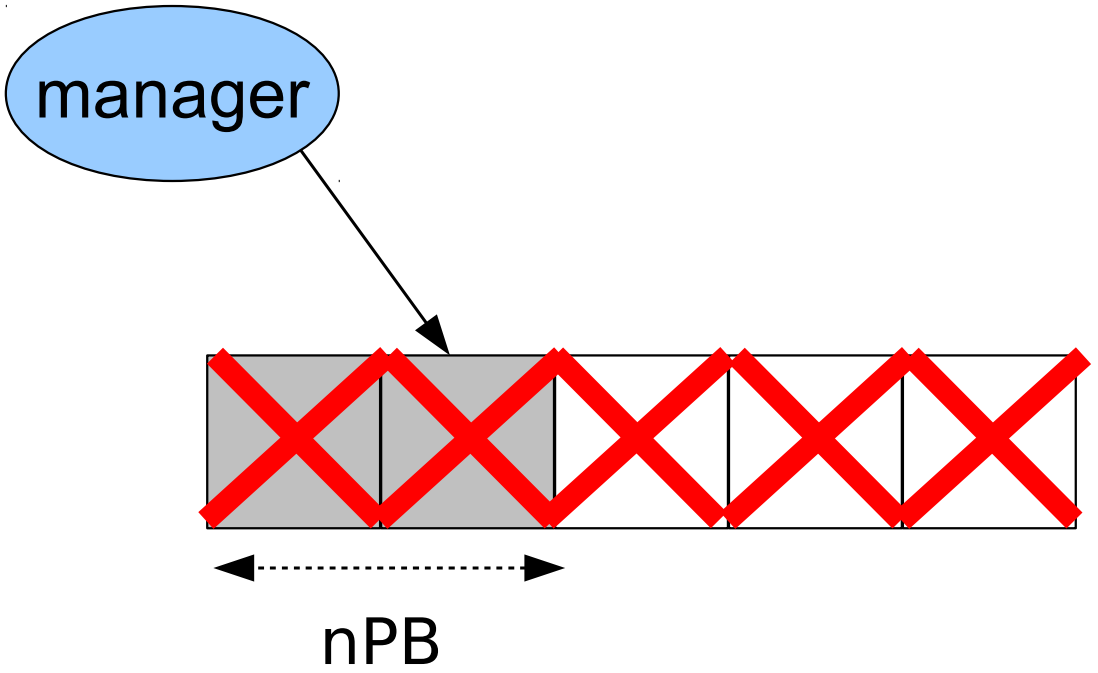


Flujo de trabajo

./exec/manager 3 2 5

nPB

[kill()]



6

Finalizar PBs

Espera PAs

Creación procesos

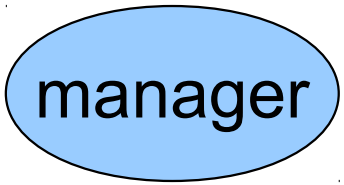
Tabla procesos

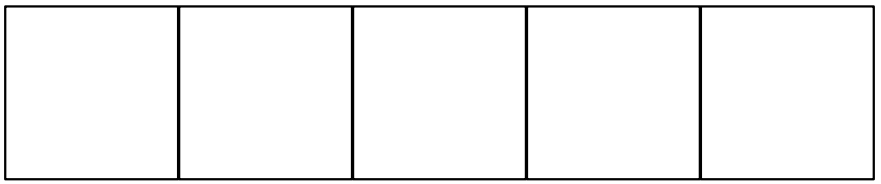
Manejador

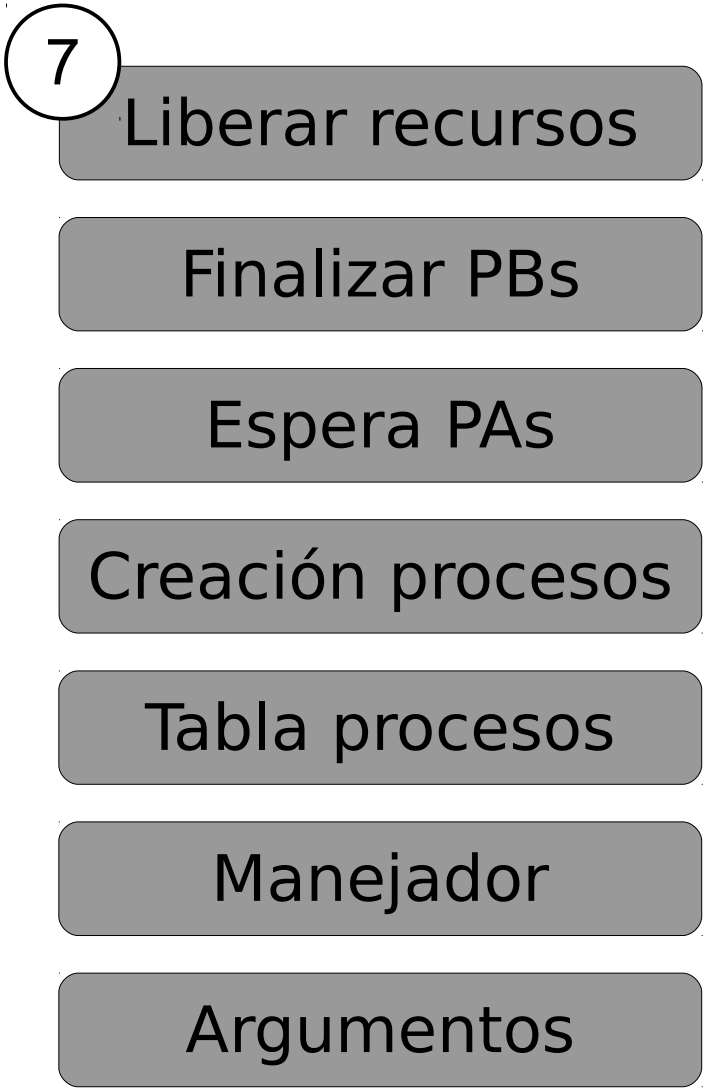
Argumentos

Flujo de trabajo

./exec/manager 3 2 5

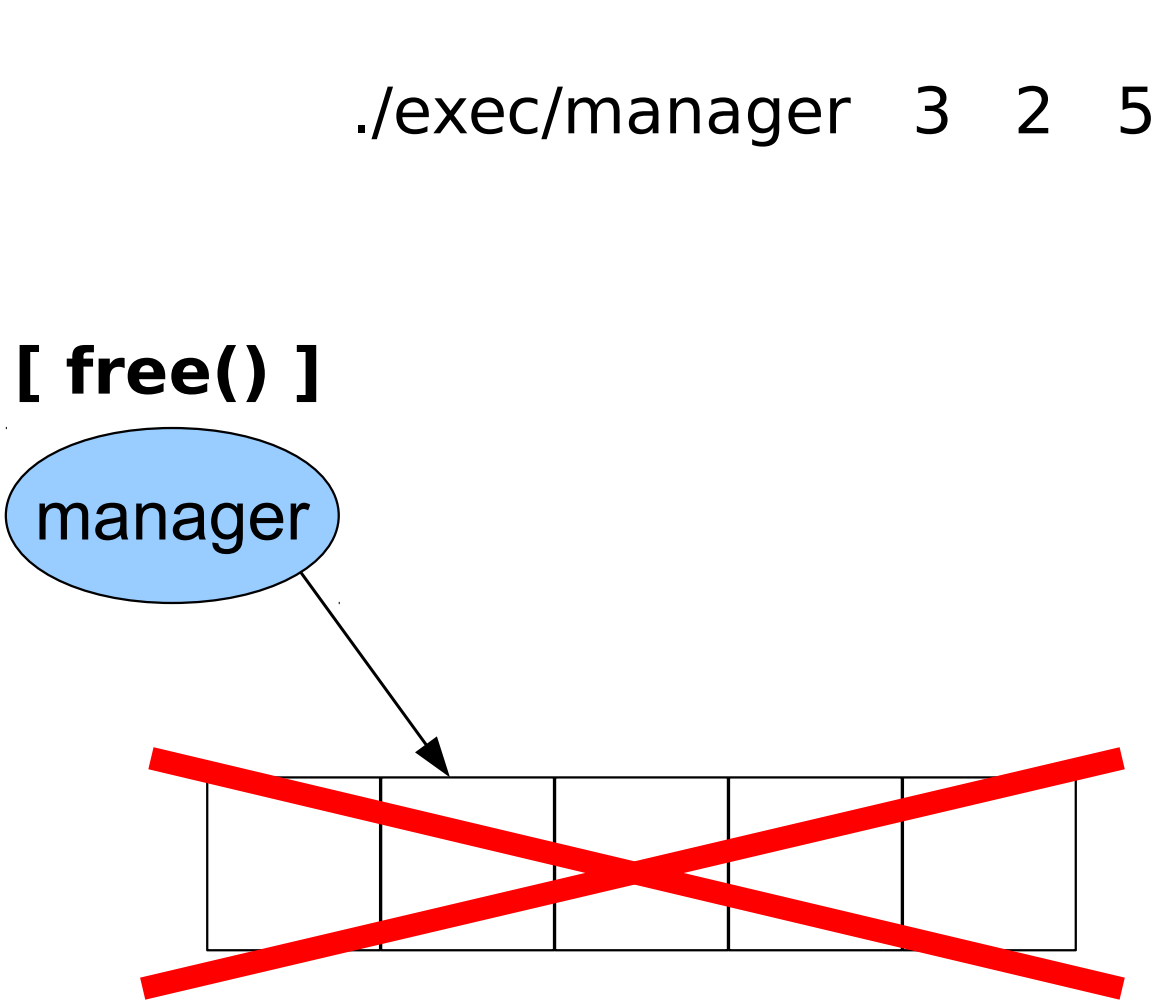






Flujo de trabajo

./exec/manager 3 2 5



7

Liberar recursos

Finalizar PBs

Espera PAs

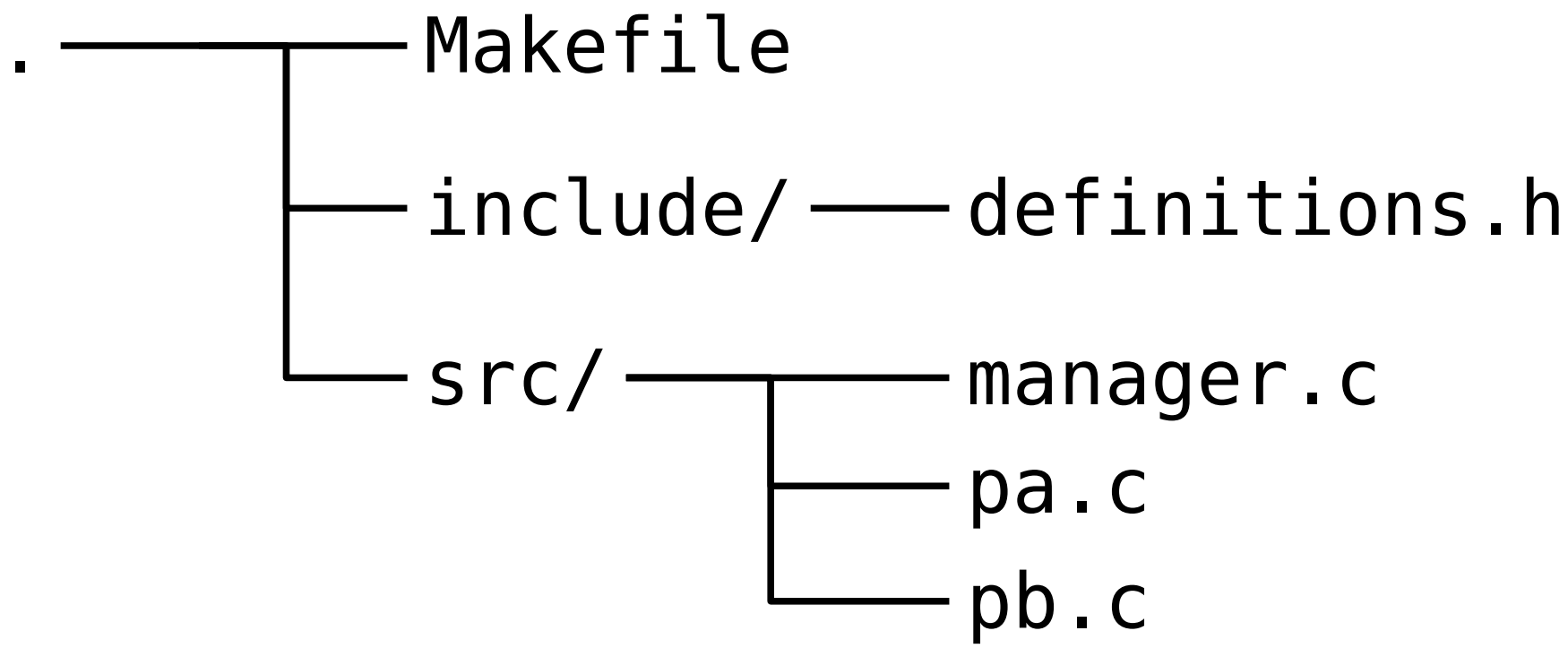
Creación procesos

Tabla procesos

Manejador

Argumentos

Código fuente



Estructuras de datos

definitions.h

```
#define PA_CLASS "PA"
#define PA_PATH "./exec/pa"
#define PB_CLASS "PB"
#define PB_PATH "./exec/pb"

enum ProcessClass_t {PA, PB};

struct TProcess_t {
  enum ProcessClass_t class; /* PA or PB */
  pid_t pid;                 /* Process PID */
  /* String repr. of the process class */
  char *str_process_class;
};
```

Estructuras de datos

manager.c

```
/* Total number of created processes */
int g_nProcesses;
/* 'Process table' (child processes) */
struct TProcess_t *g_process_table;
```

Gestión de procesos

manager.c

```
void create_processes_by_class(
  enum ProcessClass_t class, int n_new_processes,
  int index_process_table, char *s_tmax_wait);
pid_t create_single_process(
  const char *str_process_class,
  const char *path, const char *arg);
void get_str_process_info(
  enum ProcessClass_t class, char **path,
  char **str_process_class);
void init_process_table(int nPA, int nPB);
void terminate_processes(void);
void wait_processes(int nPA);
```

Funciones auxiliares

manager.c

```
void free_resources();
void install_signal_handler();
void parse_argv(
  int argc, char *argv[],
  int *nPA, int *nPB, char **s_tmax_wait);
void signal_handler(int signo);
```

Compilación automática

Makefile

```
DIROBJ := obj/
DIREXE := exec/
DIRHEA := include/
DIRSRC := src/

CFLAGS := -I$(DIRHEA) -c -Wall -ansi
LDLIBS := -lpthread -lrt
CC := gcc

all : dirs manager pa pb

dirs:
   mkdir -p $(DIROBJ) $(DIREXE)
```

Compilación automática

Makefile

```
manager: $(DIROBJ)manager.o
    $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

pa: $(DIROBJ)pa.o
    $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

pb: $(DIROBJ)pb.o
    $(CC) -o $(DIREXE)$@ $^ $(LDLIBS)

$(DIROBJ)%.o: $(DIRSRC)%.c
    $(CC) $(CFLAGS) $^ -o $@
```

Compilación automática

Makefile

```
test:
    ./$(DIREXE)manager 3 2 5

solution:
    ./$(DIREXE)manager 2 3 4

clean :
    rm -rf *~ core $(DIROBJ) $(DIREXE)
           $(DIRHEA)*~ $(DIRSRC)*~
```

Bibliografía

- Rochkind, M.J., *Advanced Unix Programm.*
  - Señales: Sec. 5.8, 5.9, 9.1, 9.2