EJERCICIOS-IMPORTANTES-para-exam...

charlieangel

Sistemas Operativos

2º Grado en Ingeniería Informática

Escuela Politécnica Superior de Córdoba
Universidad de Córdoba

Accede al documento original

pierdo espacio ⟷

Necesito concentración

ali ali oooh esto con 1 coin me lo quito yo...

WUOLAH

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <errno.h>
#include <sys/types.h>
#include <sys/wait.h>

struct datos
{
   int contlin;
   char *nom_fich;
};

void *funcionHebra(void *p)
{
   struct datos *d = (struct datos *)p;

   FILE *fich = fopen(d->nom_fich, "r");
   if (!fich)
   {
       perror("No existe el fichero\n");
       exit(EXIT_FAILURE);
   }

   char line[1024]; int palabras = 0;
   while (fgets(line, sizeof(line), fich))
   {
       d->contlin++;
   }

   fclose(fich);

   int *res = (int *)malloc(sizeof(int));
   *res = d->contlin;
```

ejercicio devuelve nº total de líneas de los ficheros pasados x arg

para contar las palabras

```c
char * p = strtok (linea, "\n\t");
while (p)
{   palabras++;
    p = strtok (NULL, "\n\t");
}
```

dejar un espacio

dejar un espacio

```
    pthread_exit(res);
}

int main(int argc, char **argv)
{
    if (argc < 2)
    {
        perror("Error en la línea de argumentos. Uso:./ej2 <nomfich1> <nomfich2> <nomfichN>\n");
        exit(EXIT_FAILURE);
    }

    struct datos x[argc - 1];
    pthread_t hilos[argc - 1];
    int *res = malloc(sizeof(int));
    int totallin = 0;

    for (int i = 0; i < (argc - 1); i++)
    {
        x[i].contlin = 0;
        x[i].nom_fich = argv[i + 1];
        if (pthread_create(&hilos[i], NULL, funcionHebra, &x[i]) != 0)
        {
            perror("Error pthread_create\n");
            exit(EXIT_FAILURE);
        }
    }
    for (int i = 0; i < (argc - 1); i++)
    {
        if (pthread_join(hilos[i], (void **)&res) != 0)
        {
            perror("Error en pthread_join\n");
            exit(EXIT_FAILURE);
        }
        totallin += *res;
    }
```

```
   printf("El total de todas las líneas de los ficheros pasados por línea de argumentos es:%d\n", totallin);
   exit(EXIT_SUCCESS);
}
```

pierdo espacio

Necesito concentración

ali ali oooh esto con 1 coin me lo quito yo...

WUOLAH

ejercicio Par e Impar

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>
#include <time.h>

// Variables globales
int par = 0;
int impar = 0;
pthread_mutex_t semp;
pthread_mutex_t semi;

void *funcionHilo(void *param)
{
   int *orden = (int *)param;
   int suma = 0;
   int numran[5];
   for (size_t i = 0; i < 5; i++)
   {
       numran[i] = rand() % 11;
       suma += numran[i];
   }
   if (*orden % 2 == 0) // orden par
   {
       if (pthread_mutex_lock(&semp) != 0)
       {
           fprintf(stderr, "Error en el lock del semp.\n");
           exit(EXIT_FAILURE);
       }
       par += suma;
       if (pthread_mutex_unlock(&semp) != 0)
       {
           fprintf(stderr, "Error en el unlock del semp.\n");
           exit(EXIT_FAILURE);
       }
```

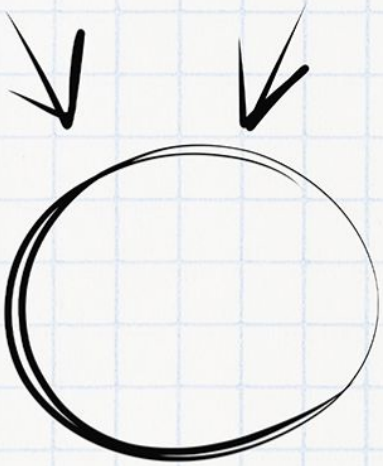
```
    }
    else if (*orden % 2 == 1) // orden impar
    {
        if (pthread_mutex_lock(&semi) != 0)
        {
            fprintf(stderr, "Error en el lock del semi.\n");
            exit(EXIT_FAILURE);
        }
        impar += suma;
        if (pthread_mutex_unlock(&semi) != 0)
        {
            fprintf(stderr, "Error en el unlock del semi.\n");
            exit(EXIT_FAILURE);
        }
    }

    int *valorDevuelto = (int *)malloc(sizeof(int));
    *valorDevuelto = suma;
    pthread_exit((void *)valorDevuelto);
}

int main(int argc, char **argv)
{
    if (argc != 2)
    {
        fprintf(stderr, "Error en la línea de argumentos\n");
        exit(EXIT_FAILURE);
    }
    if (atoi(argv[1]) <= 0)
    {
        fprintf(stderr, "Error el valor debe ser positivo y mayor a 0\n");
        exit(EXIT_FAILURE);
    }

    int N = atoi(argv[1]);
```

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

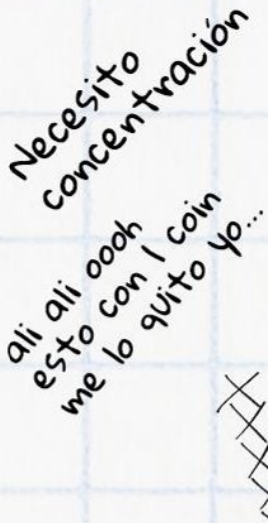
```c
    srand(time(NULL));
    int orden[N];

    // reserva de memoria para los hilos
    pthread_t *hilo = (pthread_t *)malloc(sizeof(pthread_t) * N);

    // INICIALIZACIÓN DE LOS SEMAFOROS
    if (pthread_mutex_init(&semp, NULL) != 0)
    {
        fprintf(stderr, "Error en la inicialización del semáforo par.\n");
        exit(EXIT_FAILURE);
    }
    if (pthread_mutex_init(&semi, NULL) != 0)
    {
        fprintf(stderr, "Error en la inicialización del semáforo impar.\n");
        exit(EXIT_FAILURE);
    }
    // creacion de los hilos
    for (int i = 0; i < N; i++)
    {
        orden[i] = i + 1;
        if (pthread_create(&hilo[i], NULL, (void *)funcionHilo, (void *)&orden[i]) != 0)
        {
            fprintf(stderr, "Error en la creacion del hilo.\n");
            exit(EXIT_FAILURE);
        }
    }

    int *valorDevuelto;
    // espera de los hilos
    for (int i = 0; i < N; i++)
    {
        if (pthread_join(hilo[i], (void **)&valorDevuelto))
        {
            fprintf(stderr, "Error en la espera de los hilos.\n");
```


```
            exit(EXIT_FAILURE);
        }

        printf("Main()...La hebra de orden de creación %d devolvió el valor de la suma:%d\n", i + 1, *valorDevuelto);
        free(valorDevuelto);
    }

    // DESTRUCCIÓN DE LOS SEMAFOROS
    if (pthread_mutex_destroy(&semp) != 0)
    {
        fprintf(stderr, "Error en la destruccion del semaforo par.\n");
        exit(EXIT_FAILURE);
    }
    if (pthread_mutex_destroy(&semi) != 0)
    {
        fprintf(stderr, "Error en la destruccion del semaforo impar.\n");
        exit(EXIT_FAILURE);
    }

    // Muestra de los valores de par e impar
    printf("\n");
    printf("Main()...Valor de la variable compartida impar:%d\n", impar);
    printf("Main()...Valor de la variable compartida par:%d\n", par);

    // LIBERACIÓN DE MEMORIA
    free(hilo);

    exit(EXIT_SUCCESS);
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <pthread.h>

pthread_mutex_t semA, semB;

void *escribirA(void *p)
{
   for (int i = 0; i < 5; i++)
   {
       pthread_mutex_lock(&semA);
       printf("A");
       fflush(NULL);
       pthread_mutex_unlock(&semB);
   }
}
void *escribirB(void *p)
{
   for (int i = 0; i < 5; i++)
   {
       pthread_mutex_lock(&semB);
       printf("B");
       fflush(NULL);
       pthread_mutex_unlock(&semA);
   }
}
int main()
{
   pthread_t hiloA, hiloB;

   if (pthread_mutex_init(&semA, NULL))
   {
       perror("Error en la inicializacion del semA\n");
       exit(EXIT_FAILURE);
```

```
}
if (pthread_mutex_init(&semB, NULL))
{
    perror("Error en la inicializacion del semB\n");
    exit(EXIT_FAILURE);
}

pthread_mutex_lock(&semA);

if (pthread_create(&hiloB, NULL, escribirB, NULL) != 0)
{
    perror("Error create\n");
    exit(EXIT_FAILURE);
}
if (pthread_create(&hiloA, NULL, escribirA, NULL) != 0)
{
    perror("Error create\n");
    exit(EXIT_FAILURE);
}
pthread_mutex_unlock(&semB);

if (pthread_join(hiloB, NULL) != 0)
{
    perror("Error join\n");
    exit(EXIT_FAILURE);
}
if (pthread_join(hiloA, NULL) != 0)
{
    perror("Error join\n");
    exit(EXIT_FAILURE);
}

if (pthread_mutex_destroy(&semA) == -1)
{
    perror("Error destroy\n");
```


```
        exit(EXIT_FAILURE);
    }
    if (pthread_mutex_destroy(&semB) == -1)
    {
        perror("Error destroy\n");
        exit(EXIT_FAILURE);
    }
    printf("\n");
    exit(EXIT_SUCCESS);
}
```



```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/wait.h>
#include <string.h>
#include <errno.h>

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

   if (argc < 4)
   {
       fprintf(stderr, "Error en la línea de argumentos.\n");
       exit(EXIT_FAILURE);
   }
   char *calculadora = argv[1];
   char *editortxt = argv[2];
   // Preparamos argumentos para el editor
   char **argm = malloc((argc - 2) * sizeof(char *));
   argm[0] = editortxt; // el nombre del editor debe ir primero "gedit"
   for (int i = 3; i < argc; i++)
   {
       argm[i - 2] = argv[i]; // desplazar
   }
   argm[argc - 2] = NULL; // terminador NULL
   printf("Soy el proceso con PID:%ld y PPID:%ld\n", (long int)getpid(), (long int)getppid());

   for (int i = 0; i < 2; i++)
   {
       pid_t pid;
       pid = fork();
       switch (pid)
       {
       case -1:
```

```
            fprintf(stderr, "Error al crear al hijo.\n");
            exit(EXIT_FAILURE);
        case 0:
            printf("Soy el proceso con PID:%ld y PPID:%ld\n", (long int)getpid(), (long int)getppid());
            if (i == 0)
            {                    gnome-calculator
                if (execlp(calculadora, calculadora, NULL) != -1)
                {
                    perror("Error en execlp().\n");
                    exit(EXIT_FAILURE);
                }
            }
            else if (i == 1)
            {
                if (execvp(editortxt, argm) != -1)
                {
                    perror("Error en execlp().\n");
                    exit(EXIT_FAILURE);
                }
            }
            exit(EXIT_SUCCESS);
        }
    }

    pid_t flag;
    int status;
    while ((flag = waitpid(-1, &status, 0)) > 0)
    {
        if (WIFEXITED(status))
        {
            printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long int)flag, WEXITSTATUS(status));
        }
        else if (WIFSIGNALED(status)) // Para seniales como las de finalizar o matar
        {
```


```
            printf("Proceso padre %d, hijo con PID %ld finalizado al recibir la señal %d\n", getpid(), (long int)flag,
WTERMSIG(status));
        }
    }
    if (flag == (pid_t)-1 && errno == ECHILD) // Entra cuando vuelve al while y no hay más hijos que esperar
    {
        printf("Proceso padre %d, no hay mas hijos que esperar. Valor de errno = %d, definido como: %s\n", getpid(), errno,
strerror(errno));
    }
    else
    {
        printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido como: %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    free(argm);
    return 0;
}
```

productor y consumidor

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <pthread.h>
#include <semaphore.h>

int *buffer = NULL;
int buffout = 0;
int buffin = 0;
sem_t semaforoProductor;
sem_t semaforoConsumidor;
sem_t lleno;
sem_t vacio;

void *productor(void *parametro)
{
   int *tamBuffer = (int *)parametro;

   if (sem_wait(&lleno) != 0)
   {
       fprintf(stderr, "Error en sem_wait\n");
       exit(EXIT_FAILURE);
   }

   if (sem_wait(&semaforoProductor) != 0)
   {
       fprintf(stderr, "Error en sem_wait\n");
       exit(EXIT_FAILURE);
   }

   int produce = (rand() % 10) + 1;
   printf("Soy el hilo productor %lu y produzco %d en la posición del buffer %d.\n", pthread_self(), produce, buffin);
   buffer[buffin] += produce;
   buffin = (buffin + 1) % (*tamBuffer);
```

inicializa a NULL

```
    if (sem_post(&semaforoProductor) != 0)
    {
        fprintf(stderr, "Error en sem_wait\n");
        exit(EXIT_FAILURE);
    }

    if (sem_post(&vacio) != 0)
    {
        fprintf(stderr, "Error en sem_wait\n");
        exit(EXIT_FAILURE);
    }
}

void *consumidor(void *parametro)
{
    int *tamBuffer = (int *)parametro;

    if (sem_wait(&vacio) != 0)
    {
        fprintf(stderr, "Error en sem_wait\n");
        exit(EXIT_FAILURE);
    }

    if (sem_wait(&semaforoConsumidor) != 0)
    {
        fprintf(stderr, "Error en sem_wait\n");
        exit(EXIT_FAILURE);
    }

    int consume = (rand() % 10) + 1;
    if (buffer[buffout] < consume)
    {
        consume = buffer[buffout];
    }
```


```
    printf("Soy el hilo consumidor %lu y consumo %d en la posición del buffer %d.\n", pthread_self(), consume, buffout);
    buffer[buffout] -= consume;
    buffout = (buffout + 1) % (*tamBuffer);

    if (sem_post(&semaforoConsumidor) != 0)
    {
        fprintf(stderr, "Error en sem_wait\n");
        exit(EXIT_FAILURE);
    }

    if (sem_post(&lleno) != 0)
    {
        fprintf(stderr, "Error en sem_wait\n");
        exit(EXIT_FAILURE);
    }
}

int main(int argc, char **argv)
{
    if (argc != 3)
    {
        fprintf(stderr, "Error en línea de argumentos.\n");
        exit(EXIT_FAILURE);
    }

    int N = atoi(argv[1]);
    int tamBuffer = atoi(argv[2]);
    pthread_t idhilosProductores[N];
    pthread_t idhilosConsumidores[N];
    srand(time(NULL));

    buffer = (int *)malloc(sizeof(int) * tamBuffer);
    for (int i = 0; i < tamBuffer; i++)
    {
        buffer[i] = 0;
```

```
}

if (sem_init(&semaforoProductor, 0, 1) != 0)
{
    fprintf(stderr, "Error en sem_init\n");
    exit(EXIT_FAILURE);
}

if (sem_init(&semaforoConsumidor, 0, 1) != 0)
{
    fprintf(stderr, "Error en sem_init\n");
    exit(EXIT_FAILURE);
}

if (sem_init(&lleno, 0, tamBuffer) != 0)
{
    fprintf(stderr, "Error en sem_init\n");
    exit(EXIT_FAILURE);
}

if (sem_init(&vacio, 0, 0) != 0)
{
    fprintf(stderr, "Error en sem_init\n");
    exit(EXIT_FAILURE);
}

for (size_t i = 0; i < N; i++)
{
    if (pthread_create(&idhilosProductores[i], NULL, (void *)productor, (void *)&tamBuffer) != 0)
    {
        fprintf(stderr, "Error en pthread_create()\n");
        exit(EXIT_FAILURE);
    }
}
```

```
for (size_t i = 0; i < N; i++)
{
    if (pthread_create(&idhilosConsumidores[i], NULL, (void *)consumidor, (void *)&tamBuffer) != 0)
    {
        fprintf(stderr, "Error en pthread_create()\n");
        exit(EXIT_FAILURE);
    }
}

for (size_t i = 0; i < N; i++)
{
    if (pthread_join(idhilosProductores[i], (void **)NULL) != 0)
    {
        fprintf(stderr, "Error en pthread_join()\n");
        exit(EXIT_FAILURE);
    }
}

for (size_t i = 0; i < N; i++)
{
    if (pthread_join(idhilosConsumidores[i], (void **)NULL) != 0)
    {
        fprintf(stderr, "Error en pthread_join()\n");
        exit(EXIT_FAILURE);
    }
}

if (sem_destroy(&semaforoProductor) != 0)
{
    fprintf(stderr, "Error en sem_destroy\n");
    exit(EXIT_FAILURE);
}

if (sem_destroy(&semaforoConsumidor) != 0)
{
```

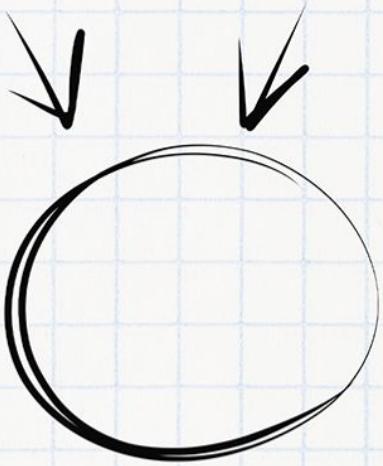

```
        fprintf(stderr, "Error en sem_destroy\n");
        exit(EXIT_FAILURE);
    }

    if (sem_destroy(&lleno) != 0)
    {
        fprintf(stderr, "Error en sem_destroy\n");
        exit(EXIT_FAILURE);
    }

    if (sem_destroy(&vacio) != 0)
    {
        fprintf(stderr, "Error en sem_destroy\n");
        exit(EXIT_FAILURE);
    }

    printf("El buffer al finalizar el programa es el siguiente:\n");
    for (size_t i = 0; i < tamBuffer; i++)
    {
        printf("%d ", buffer[i]);
    }
    printf("\n");

    free(buffer);

    exit(EXIT_SUCCESS);
}
```

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

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>

void funcionManejadoraHijo(int signal)
{
   printf("Soy el proceso %ld y he recibido la señal SIGUSR1(%d)...\n", (long int)getpid(), signal);
}

int main(int argc, char **argv)
{
   if (argc != 2)
   {
       fprintf(stderr, "Error en línea de argumentos.\n");
       exit(EXIT_FAILURE);
   }

   for (size_t i = 0; i < 2; i++)
   {
       pid_t pid = fork();
       switch (pid)
       {
       case -1:
           fprintf(stderr, "Error en fork().\n");
           exit(EXIT_FAILURE);

       case 0:
           printf("Soy el proceso hijo con PID %ld y PPID %ld.\n", (long int)getpid(), (long int)getppid());
           if (i == 0)
           {
               if (execlp("ls", "ls", NULL) == -1)
```

```
        {
            fprintf(stderr, "Error en execlp.\n");
            exit(EXIT_FAILURE);
        }
        exit(EXIT_SUCCESS);
    }
    else
    {
        pid_t pid1 = fork();
        switch (pid1)
        {
        case -1:
            fprintf(stderr, "Error en fork().\n");
            exit(EXIT_FAILURE);
        case 0:
            signal(SIGUSR1, &funcionManejadoraHijo);
            while (1)
            {
                pause();
            }
        }
        pid_t pid2 = fork();
        switch (pid2)
        {
        case -1:
            fprintf(stderr, "Error en fork().\n");
            exit(EXIT_FAILURE);
        case 0:
            signal(SIGUSR1, &funcionManejadoraHijo);
            while (1)
            {
                pause();
            }
        }
```

nieto1

nieto2


```
            sleep(1);
            for (int i = 0; i < atoi(argv[1]); i++)
            {
                kill(pid1, SIGUSR1);
                kill(pid2, SIGUSR1);
                sleep(1);
            }

            kill(pid1, SIGKILL);
            kill(pid2, SIGKILL);
        }
    }
}

int status;
pid_t flag;
while ((flag = waitpid(-1, &status, 0)) > 0)
{
    if (WIFEXITED(status))
    {
        printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long int)flag, WEXITSTATUS(status));
    }
    else if (WIFSIGNALED(status)) // Para seniales como las de finalizar o matar
    {
        printf("Proceso padre %d, hijo con PID %ld finalizado al recibir la señal %d\n", getpid(), (long int)flag,
WTERMSIG(status));
    }
}
if (flag == (pid_t)-1 && errno == ECHILD) // Entra cuando vuelve al while y no hay más hijos que esperar
{
    printf("Proceso padre %d, no hay mas hijos que esperar. Valor de errno = %d, definido como: %s\n", getpid(), errno,
strerror(errno));
}
else
```

hijo 2 → padre de los nietos

y envíar SIGUSR1 a nieto1 y nieto2

y los mata

exit(EXIT_SUCCESS);

```
    {
        printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido como: %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

version sin sleep()

```c
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <errno.h>
#include <sys/wait.h>
#include <unistd.h>
#include <signal.h>
void funcionManejadoraHijo(int signal)
{
   printf("Soy el proceso %ld y he recibido la señal SIGUSR1(%d)...\n", (long int)getpid(), signal);
   kill(getppid(), SIGUSR1);
}

void funcionManejadoraPadre(int signal)
{
}

int main(int argc, char **argv)
{
   if (argc != 2)
   {
       fprintf(stderr, "Error en línea de argumentos.\n");
       exit(EXIT_FAILURE);
   }

   for (size_t i = 0; i < 2; i++)
   {
       pid_t pid = fork();
       switch (pid)
       {
       case -1:
           fprintf(stderr, "Error en fork().\n");
           exit(EXIT_FAILURE);
```

pierdo espacio

Necesito concentración

ali ali oooh esto con 1 coin me lo quito yo...

WUOLAH

```
        case 0:
            printf("Soy el proceso hijo con PID %ld y PPID %ld.\n", (long int)getpid(), (long int)getppid());
            if (i == 0)
            {
                if (execlp("ls", "ls", NULL) == -1)
                {
                    fprintf(stderr, "Error en exexlp.\n");
                    exit(EXIT_FAILURE);
                }
                exit(EXIT_SUCCESS);
            }
            else
            {
                signal(SIGUSR1, &funcionManejadoraPadre);
                pid_t pid1 = fork();
                switch (pid1)
                {
                case -1:
                    fprintf(stderr, "Error en fork().\n");
                    exit(EXIT_FAILURE);
                case 0:
                    signal(SIGUSR1, &funcionManejadoraHijo);
                    kill(getppid(), SIGUSR1);
                    while (1)
                    {
                        pause();
                    }
                }
                pause();
                pid_t pid2 = fork();
                switch (pid2)
                {
                case -1:
                    fprintf(stderr, "Error en fork().\n");
                    exit(EXIT_FAILURE);
```

configuramos

desbloquear padre

nieto 1

```
            case 0:
                signal(SIGUSR1, &funcionManejadoraHijo);
                kill(getppid(), SIGUSR1);
                while (1)
                {
                    pause();
                }
            }
        //el padre
            pause();

            for (int i = 0; i < atoi(argv[1]); i++)
            {
                kill(pid1, SIGUSR1);
                pause();
                kill(pid2, SIGUSR1);
                pause();
            }

            kill(pid1, SIGKILL);
            kill(pid2, SIGKILL);
        }
    }
}

int status;
pid_t flag;
while ((flag = waitpid(-1, &status, 0)) > 0)
{
    if (WIFEXITED(status))
    {
        printf("Proceso padre %d, hijo con PID %ld finalizado, status = %d\n", getpid(), (long int)flag, WEXITSTATUS(status));
    }
    else if (WIFSIGNALED(status)) // Para seniales como las de finalizar o matar
    {
```

nieto2

hijo2 (padre de nietos)

exit(exit_success);

```
            printf("Proceso padre %d, hijo con PID %ld finalizado al recibir la señal %d\n", getpid(), (long int)flag,
WTERMSIG(status));
        }
    }
    if (flag == (pid_t)-1 && errno == ECHILD) // Entra cuando vuelve al while y no hay más hijos que esperar
    {
        printf("Proceso padre %d, no hay mas hijos que esperar. Valor de errno = %d, definido como: %s\n", getpid(), errno,
strerror(errno));
    }
    else
    {
        printf("Error en la invocacion de wait o waitpid. Valor de errno = %d, definido como: %s\n", errno, strerror(errno));
        exit(EXIT_FAILURE);
    }

    exit(EXIT_SUCCESS);
}
```

ejercicio con alarm()

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>

int cont = 0;

void tratarAlarma(int s)
{
   printf("RING--> %d \n", cont + 1);
   cont++;
}

int main(void)
{
   // configuramos lo que debe hacer el proceso cuando reciba una alarm
   signal(SIGALRM,&tratarAlarma);

   // PRIMERA ALARMA --> 5segs
   alarm(5);

   // SEGUNDA ALARMA --> 3 segs
   if (cont == 1)
   {
       alarm(3);
   }
   // SIGUIENTES ALARMAS (hasta 4)--> cada segundo
   while (1)
   {
       pause();
       alarm(1);
       if (cont == 4)
       {
           kill(getpid(), SIGKILL);
       }
```

```
    }

    return 0;
}
```