trabajos_universidad/README.md at master · germaaan/trabajos_universidad · GitHub

```
<meta content="https://avatars3.githubusercontent.com/u/5518719?s=400&amp;v=4"
property="og:image" /><meta content="GitHub" property="og:site_name" /><meta
content="object" property="og:type" /><meta content="germaaan/trabajos_universidad"
property="og:title" /><meta
content="https://github.com/germaaan/trabajos_universidad" property="og:url"
/><meta content="trabajos_universidad - Recopilación de todos los trabajos
(recuperables) hechos durante mi trayecto en la Universidad de Granada"
property="og:description" />
```

```
<meta name="google-site-verification"
content="KT5gs8h0wvaagLKAVWq8bbeNwnZZK1r1XQysX3xurLU">
```

```
  <meta name="hostname" content="github.com">
```

```
  <meta name="expected-hostname" content="github.com">
<meta name="js-proxy-site-detection-payload"
content="ODU0ZmNhZWMwYTQ0ZjNiMGJhZjNjNGUyZThlZmViMzI5M2I5YTYzODc4ZDc0Yzk5YzlkM2NkM2
VlNTk2NzMwZHx7InJlbW90ZV9hZGRyZXNzIjoiNzkuMTUwLjIxNi4xOTgiLCJyZXF1ZXN0X2lkIjoiRkIyM
ToyOTAzMTo0MkY3QkIxOjZCMjRBMjY6NUEzOTFDODQiLCJ0aW1lc3RhbXAiOjE1MTM2OTIyOTMsImhvc3Qi
OiJnaXRodWIuY29tIn0=">
```

```
  <link href="https://github.com/germaaan/trabajos_universidad/commits/master.atom"
rel="alternate" title="Recent Commits to trabajos_universidad:master"
type="application/atom+xml">
```

```
<link rel="canonical"
href="https://github.com/germaaan/trabajos_universidad/blob/master/2GII/SO/practica
_02/README.md" data-pjax-transient>
```

germaaan/trabajos_universidad

trabajos_universidad/2GII/SO/practica_02/**README.md**

720 lines (535 sloc) 19.9 KB

Sistemas Operativos

2º Grado en Ingeniería Informática 2011/2012

Práctica 2: Programación de la API (Application Programming Interface) de Linux.

Germán Martínez Maldonado

Sesión 1: Gestión y comunicación de procesos (Parte I)

Actividad 1: Dos estructuras de procesos muy frecuentes son las que se muestran en la Figura 3. Crear un programa que cree cada una de las estructuras con tres procesos (un padre y dos hijos). Intentad que el código sea lo más compacto posible (pista: fork() debería forma parte de un bucle que se ejecuta tantas veces como procesos quiero crear, sin contar al padre).

Ventilador de procesos:

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <cstdlib>

using namespace std;

int main(int argc, char **argv) {
    int n;
    pid_t pid_hijo, pid_padre;

    if (argc != 2) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <numeroProcesosHijo>\n";
        exit(1);
    }

    n = atoi(argv[1]);
    pid_padre = getpid();

    cout << "PID del padre: " << pid_padre << endl;

    pid_hijo = fork();

    for (int i = 0; i < n; i++) {

        switch (pid_hijo) {
            case (pid_t) - 1:
                perror("fork fallo"); /* fork ha fallado */
                break;

            case (pid_t) 0: /* en el hijo pid_hijo == */
                break;

            default:
                pid_padre = getpid();
```

```
                pid_hijo = fork();
        }
    }

    cout << "Hijo creado con PID: " << getpid() << ", PID del padre " << pid_padre
<< endl;

    return 0;
}
```

Línea de procesos:

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <cstdlib>

using namespace std;

int main(int argc, char **argv) {
    int n, estado;
    pid_t pid_hijo, pid_padre;

    if (argc != 2) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <numeroProcesosHijo>\n";
        exit(1);
    }

    n = atoi(argv[1]);
    pid_padre = getpid();

    cout << "PID del padre: " << pid_padre << endl;

    pid_hijo = fork();

    for (int i = 0; i < n; i++) {

        switch (pid_hijo) {
            case (pid_t) - 1:
                perror("fork fallo"); /* fork ha fallado */
                break;

            case (pid_t) 0: /* en el hijo pid_hijo == */
                pid_padre = getpid();
                pid_hijo = fork();
                break;
        }
    }

    cout << "Hijo creado con PID: " << getpid() << ", PID del padre " << pid_padre
<< endl;

    return 0;
}
```

Actividad 2: Debéis construir tres programas, denominados lanzador, imp_c e imp_i. Estos programas realizan las siguientes funciones:

- imp_c imprime 30 veces un carácter alfabético que se le pasa como argumento. Si el carácter que se le pasa es numérico, retorna con error.
- imp_i: imprime 20 veces un carácter numérico pasado como argumento. De forma análoga, si le pasamos un carácter alfabético devuelve error.
- lanzador: crea dos procesos hijos, una para ejecutar imp_c y otro para ejecutar imp_i de forma concurrente. El proceso espera a que los dos hijos finalicen y comprueba el estado de finalización de cada uno, indicándolo en pantalla.

De esta actividad haremos dos variantes, la primera con fork() y la segunda con clone(). En el segundo caso, en lugar de ser procesos independientes, los procesos hijos pueden ser hilos del programa padre, es decir, tendremos un main() con dos funciones imp_c() e imp_i().

Versión fork:

- imp_c:

```
#include <iostream>

using namespace std;

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

    char c = *argv[1];

    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {

        for (int i = 0; i < 30; i++)
            cout << c << " ";
    } else {
        cout << "Error: se esperaba un carácter alfabético.";
    }

    cout << endl << flush;

    return 1;
}
```

- imp_i:

```
#include <iostream>
#include <cstdlib>

using namespace std;

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

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

```
    if (i > 0 && i <= 9) {

        for (int j = 0; j < 30; j++)
            cout << i << " ";
    } else {
        cout << "Error: se esperaba un valor numérico.";
    }

    cout << endl << flush;

    return 2;
}
```

- lanzador:

```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <sys/syscall.h>

using namespace std;

int main(int argc, char **argv) {
    pid_t pid, pid1, pid2;
    int estado1, estado2, ejecucion1, ejecucion2;
    char c, i;

    if (argc != 3) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <caracter> <entero>\n";
        exit(1);
    }

    c = *argv[1];
    i = *argv[2];

    switch (pid1 = fork()) {
        case (pid_t) - 1:
            perror("fork fallo");
            return -1;

        case (pid_t) 0:
            cout << "\nProceso hijo 1 (PID: " << getpid() << ", PID Padre: " <<
getppid() << ")." << endl;
            cout << "Imprimiendo carácter alfabético recibido..." << endl;
            ejecucion1 = execl("./imp_c", "imp_c", &c, 0);

            if (ejecucion1 == -1) {
                perror("execl fallo");
                return -1;
            } else
                _exit(1);

        default:
```

```
            pid = waitpid(pid1, &estado1, WUNTRACED);
    }

    switch (pid2 = fork()) {
        case (pid_t) - 1:
            perror("fork fallo");
            return -1;

        case (pid_t) 0:
            cout << "\nProceso hijo 2 (PID: " << getpid() << ", PID Padre: " <<
getppid() << ")." << endl;
            cout << "Imprimiendo valor numérico recibido..." << endl;
            ejecucion2 = execl("./imp_i", "imp_i", &i, 0);

            if (ejecucion2 == -1) {
                perror("execl fallo");
                return -1;
            } else
                _exit(2);

        default:
            pid = waitpid(pid2, &estado2, WUNTRACED);
    }

    cout << "\nProceso padre (PID: " << getpid() << ")." << endl;

    if (WIFEXITED(estado1))
        cerr << pid1 << " finalizó: " << WEXITSTATUS(estado1) << endl;
    else if (WIFSTOPPED(estado1))
        cerr << pid1 << " parado por " << WSTOPSIG(estado1) << endl;
    else if (WIFSIGNALED(estado1))
        cerr << pid1 << " matado por " << WTERMSIG(estado1) << endl;
    else perror("En waitpid");

    if (WIFEXITED(estado2))
        cerr << pid2 << " finalizó: " << WEXITSTATUS(estado2) << endl;
    else if (WIFSTOPPED(estado2))
        cerr << pid2 << " parado por " << WSTOPSIG(estado2) << endl;
    else if (WIFSIGNALED(estado2))
        cerr << pid2 << " matado por " << WTERMSIG(estado2) << endl;
    else perror("En waitpid");

    return 0;
}
```

Versión fork:

```
#include <iostream>
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <sys/wait.h>
#include <sys/syscall.h>

using namespace std;
```

```
int i;
char c;

int imp_c(void *p) {
    int tid = syscall(SYS_gettid);

    cout << "\nProceso hijo 1 (PID: " << getpid() << ", TID: " << tid << ", PID
Padre: " << getppid() << ")." << endl;

    if ((c >= 'A' && c <= 'Z') || (c >= 'a' && c <= 'z')) {

        for (int i = 0; i < 30; i++)
            cout << c << " ";
    } else {
        cout << "Error: se esperaba un carácter alfabético.";
    }

    cout << endl << flush;
    _exit(1);
}

int imp_i(void *p) {
    int *i = (int*) p;
    int tid = syscall(SYS_gettid);

    cout << "\nProceso hijo 2 (PID: " << getpid() << ", TID: " << tid << ", PID
Padre: " << getppid() << ")." << endl;

    if (*i >= 0 && *i <= 9) {

        for (int j = 0; j < 30; j++)
            cout << *i << " ";
    } else {
        cout << "Error: se esperaba un valor numérico.";
    }

    cout << endl << flush;
    _exit(2);
}

int main(int argc, char **argv) {
    unsigned char pila[6144];
    int ejecucion1, ejecucion2;

    if (argc != 3) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <caracter> <entero>\n";
        exit(1);
    }

    c = *argv[1];
    i = atoi(argv[2]);

    cout << "\nProceso padre (PID: " << getpid() << ")." << endl;

    ejecucion1 = clone(imp_c, (void **) pila + 2048, CLONE_VM | CLONE_FILES |
CLONE_FS | CLONE_THREAD | CLONE_SIGHAND, NULL);
```

```
    ejecucion2 = clone(imp_i, (void **) pila + 4096, CLONE_VM | CLONE_FILES |
CLONE_FS | CLONE_THREAD | CLONE_SIGHAND, NULL);

    return 0;
}
```

Actividad 3: Consideramos un anillo de procesos a un conjunto de n procesos que se comunican a través de n cauces de forma que el proceso i-ésimo está conectado con el proceso i-1 a través del cauce i, y al proceso i+1 con el cauce i+1, tal como muestra la Figura 4. En la actividad, debéis crear un anillo de tres procesos (con la jerarquía de procesos que estiméis oportuna) que se conecten a través de tres cauces.

Creada la infraestructura de procesos y cauces indicada, el proceso padre (1), deberá leer de teclado un número, tras lo cual lo escribe en el cauce_1. Este número lo recoge el hijo (2) que lo multiplica por 2, y el resultado lo escribe en cauce_2. De forma similar se procede en proceso (3), pero este multiplica lo leído por tres, y los escribe en cauce_3. Finalmente el padre leer de cauce_3 y lo imprime en la pantalla.

```
#include <iostream>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <unistd.h>

using namespace std;

int main() {
    pid_t pid_1, pid_2;
    int cauce_1[2], cauce_2[2], cauce_3[2], estado_1, estado_2, valor_1, valor_2,
valor_3;
    char bufer[20];

    if (pipe(cauce_1) == -1) perror("cauce_1"), exit(1); /*creamos el cauce_1 */

    if (pipe(cauce_2) == -1) perror("cauce_2"), exit(2); /*creamos el cauce_2 */

    if (pipe(cauce_3) == -1) perror("cauce_2"), exit(3); /*creamos el cauce_3 */

    cout << "El padre debe leer un valor desde teclado: ";
    cin >> valor_1;

    sprintf(bufer, "%d", valor_1);
    write(cauce_1[1], bufer, sizeof (bufer));
    close(cauce_1[1]);

    switch (pid_1 = fork()) {
        case -1:
            perror("fork");
```

```
            exit(4);

        case 0:
            close(cauce_1[1]);

            read(cauce_1[0], bufer, 20);

            valor_2 = atoi(bufer) * 2;

            sprintf(bufer, "%d", valor_2);
            write(cauce_2[1], bufer, sizeof (bufer));
            close(cauce_2[1]);
            break;

        default:
            switch (pid_2 = fork()) {
                case -1:
                    perror("fork");
                    exit(5);

                case 0:
                    close(cauce_2[1]);

                    read(cauce_2[0], bufer, 20);

                    valor_3 = atoi(bufer) * 3;

                    sprintf(bufer, "%d", valor_3);
                    write(cauce_3[1], bufer, sizeof (bufer));
                    close(cauce_3[1]);
                    exit(0);
            }

            exit(0);
    }

    close(cauce_3[1]);

    read(cauce_3[0], bufer, 20);
    cout << "Padre: \"Mi segundo hijo ha escrito " << bufer << "\"" << endl;

    if (waitpid(pid_1, &estado_1, 0) == pid_1 && WIFEXITED(estado_1))
        if (waitpid(pid_2, &estado_2, 0) == pid_2 && WIFEXITED(estado_2))
            return WEXITSTATUS(estado_2);

    return 6;
}
```

Sesión 2: Gestión y comunicación de procesos (Parte II)

Actividad 1: Escriba un programa que verifique el efecto de los indicadores SA_RESETHAND y SA_NODEFER cuando se establece un manejador de señales con sigaction().

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
using namespace std;

void manejador(int sig_num) {
    cout << "Atrapada la señal: " << sig_num << endl;
}

int main(int argc, char* argv[]) {
    sigset_t sigmask;
    struct sigaction accion, vieja_accion;
    sigemptyset(&sigmask);

    /* inicializamos la máscara */
    if (sigaddset(&sigmask, SIGTERM) == -1 || sigprocmask(SIG_SETMASK, &sigmask, 0)
== -1)
        perror("Al establecer la máscara");

    sigemptyset(&accion.sa_mask);
    sigaddset(&accion.sa_mask, SIGSEGV);

    if (sigismember(&sigmask, SIGINT)) {
        accion.sa_handler = manejador;
        accion.sa_flags = SA_NODEFER | SA_RESETHAND;
        if (sigaction(SIGINT, &accion, &vieja_accion) == -1)
            perror("Sigaction");
    }

    pause();

    return 0;
}
```

Actividad 2: Implementar la función abort().

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

using namespace std;

void abortar(int) {
}

int main(int argc, char** argv) {
    struct sigaction accion;

    accion.sa_handler = abortar;
    accion.sa_flags = 0;
    sigemptyset(&accion.sa_mask);
```

```
    if (sigaction(SIGABRT, &accion, 0) == -1) {
        perror("Sigaction");
    }

    (void) abort();

    return 0;
}
```

Actividad 3: El kernel tiene un tratamiento especial con SIGCONT: si un proceso está actualmente parado, la llegada de una señal SIGCONT siempre causa que se reanude el proceso, incluso si está bloqueando o ignorando esta señal. Es más, un proceso parado que ha bloqueado SIGCONT y tiene un manejador para dicha señal, cuando se reanude al recibir la señal, el manejador solo se invocará cuando se desbloquee la señal. Hacer un programa que demuestre este comportamiento.

Nota: Recordar que podemos parar un proceso con y podemos enviarle la señal de continuación con kill -CONT (o implícitamente con la orden del shell fg).

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <signal.h>
using namespace std;

void manejador(int sig_num) {
    cout << "Proceso previamente interrumpido" << endl;
}

int main(int argc, char* argv[]) {
    sigset_t sigmask;
    struct sigaction accion, vieja_accion;
    sigemptyset(&sigmask);

    /* inicializamos la máscara */
    if (sigaddset(&sigmask, SIGTERM) == -1 || sigprocmask(SIG_SETMASK, &sigmask, 0)
== -1)
        perror("Al establecer la máscara");

    sigemptyset(&accion.sa_mask);
    sigaddset(&accion.sa_mask, SIGSEGV);

    if (sigismember(&sigmask, SIGINT)) {
        accion.sa_handler = manejador;
        accion.sa_flags = SA_NODEFER | SA_RESETHAND;
        if (sigaction(SIGCONT, &accion, &vieja_accion) == -1)
            perror("Sigaction");
    }

    raise(SIGSTOP);

    return 0;
}
```

Sesión 3: Archivos y directorios

Actividad 1: Implementar un programa que realice la misma función que cp, es decir, que admita como argumentos dos nombres de archivos, y copie el contenido de uno en el otro. El algoritmo será sencillo: abrimos el archivo origen de lectura y el archivo destino de escritura (sino existe los creamos); vamos leyendo secuencialmente el archivo origen y escribimos lo leído en el archivo destino, mientras que no alcancemos el fin de archivo; cerramos los archivos.

```
#include <sys/stat.h>
#include <fcntl.h>
#include <cstdlib>
#include <iostream>

using namespace std;

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

    if (argc != 3) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <archivoEntrada> <archivoSalida>\n";
        exit(1);
    }

    const int TAM_BUFFER = 512;
    int entrada, salida, leidos, escritos;
    char buffer[TAM_BUFFER];

    if ((entrada = open(argv[1], O_RDONLY)) == -1) {
        cerr << "Error: No se ha podido abrir el archivo \"" << argv[1] << "\" para
solo lectura.\n";
        exit(-1);
    }

    if ((salida = open(argv[2], O_WRONLY | O_CREAT | O_TRUNC)) == -1) {
        cerr << "Error: No se pudo abrir o crear el archivo \"" << argv[2] << "\"
para solo escritura.\n";
        close(entrada);
        exit(-1);
    }

    while ((leidos = read(entrada, &buffer, sizeof (buffer))) > 0) {
        if (leidos == -1) {
            cerr << "Error: Fallo al leer desde archivo \"" << argv[1] << "\".\n";
            close(entrada);
            close(salida);
            exit(-1);
        } else if (leidos > 0) {
            if ((escritos = write(salida, &buffer, leidos)) == -1) {
                cerr << "Error: Fallo al escribir en archivo \"" << argv[2] <<
"\".\n";
                close(entrada);
```

```
                close(salida);
                exit(-1);
            }
        }
    }

    if (close(entrada) == -1)
        cerr << "Error: No se ha podido cerrar correctamente el archivo \"" <<
argv[1] << "\".";

    if (close(salida) == -1)
        cerr << "Error: No se ha podido cerrar correctamente el archivo \"" <<
argv[2] << "\".";

    return (0);
}
```

Actividad 2: Construir un programa que admita como argumento el nombre de un archivo y nos indique de que tipo es y su tamaño.

```
#include <sys/stat.h>
#include <fcntl.h>
#include <cstdlib>
#include <iostream>

using namespace std;

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

    if (argc != 2) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <archivoConsulta>\n";
        exit(1);
    }

    struct stat statbuf;

    if (stat(argv[1], &statbuf) == -1) {
        cerr << "Error: No se han podido obtener los atributos del archivo \"" <<
argv[1] << "\".\n";
        exit(-1);
    } else {
        cout << "\nNombre:\t" << argv[1] << endl;

        cout << "Tipo:\t";

        switch (statbuf.st_mode & S_IFMT) {
            case (S_IFREG):
                cout << "Archivo regular" << endl;
                break;
            case (S_IFDIR):
                cout << "Directorio" << endl;
                break;
            case (S_IFCHR):
                cout << "Dispositivo de caracteres" << endl;
```

```
                break;
            case (S_IFBLK):
                cout << "Dispositivo de bloques" << endl;
                break;
            case (S_IFIFO):
                cout << "FIFO o cauce" << endl;
                break;
            case (S_IFSOCK):
                cout << "Socket" << endl;
                break;
            case (S_IFLNK):
                cout << "Enlace simbólico" << endl;
                break;
        }

        cout << "Tamaño:\t" << statbuf.st_size << " Bytes" << endl;
    }

    return (0);
}
```

Actividad 3: Construya un programa que imprima el nombre de los archivos del directorio que se le pasa como argumento e indique de tipo son.

```
#include <sys/stat.h>
#include <fcntl.h>
#include <dirent.h>
#include <cstdlib>
#include <iostream>

using namespace std;

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

    if (argc != 2) {
        cerr << "Error: Numero de parametros erroneo.\n";
        cerr << "Uso: " << argv[0] << " <directorioConsulta>\n";
        exit(1);
    }

    DIR *flujo_directorio;
    struct dirent *entrada;
    struct stat statbuf;
    int archivo;

    if ((flujo_directorio = opendir(argv[1])) == NULL) {
        cerr << "Error: No se ha podido abrir el directorio \"" << argv[1] <<
"\".\n";
        exit(1);
    }

    while (entrada = readdir(flujo_directorio)) {

        if (stat(entrada->d_name, &statbuf) == -1) {
            cerr << "Error: No se han podido obtener los atributos del archivo \""
```

```
<< argv[1] << "\".\n";
            exit(-1);
        } else {
            cout << "\nNombre:\t" << entrada->d_name << endl;

            cout << "Tipo:\t";

            switch (statbuf.st_mode & S_IFMT) {
                case (S_IFREG):
                    cout << "Archivo regular" << endl;
                    break;
                case (S_IFDIR):
                    cout << "Directorio" << endl;
                    break;
                case (S_IFCHR):
                    cout << "Dispositivo de caracteres" << endl;
                    break;
                case (S_IFBLK):
                    cout << "Dispositivo de bloques" << endl;
                    break;
                case (S_IFIFO):
                    cout << "FIFO o cauce" << endl;
                    break;
                case (S_IFSOCK):
                    cout << "Socket" << endl;
                    break;
                case (S_IFLNK):
                    cout << "Enlace simbólico" << endl;
                    break;
            }
        }
    }

    if ((closedir(flujo_directorio)) == -1)
        cerr << "Error: No se podido cerrar correctamente el directorio \"" <<
argv[1] << "\".\n";

    return (0);
}
```

- Training
- Shop
- Blog
- About

⚠ ✕ You can't perform that action at this time.

```
<script crossorigin="anonymous"
integrity="sha256-OQfZqtgSuMlBR8iBZctKlpPFfPmfDo4BBT/Hneep2RE="
src="https://assets-cdn.github.com/assets/frameworks-3907d9aad812b8c94147c88165cb4a
9693c57cf99f0e8e01053fc79de7a9d911.js"></script>

<script async="async" crossorigin="anonymous"
integrity="sha256-H3USKiPnG1c63j+5wLGbfVlH6B7Z13A4ul8HC8CnSLQ="
src="https://assets-cdn.github.com/assets/github-1f75122a23e71b573ade3fb9c0b19b7d59
47e81ed9d77038ba5f070bc0a748b4.js"></script>
```

⚠ You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session.