



Universidad Veracruzana



Ejercicio 3

Alumno(s):

David Barcenás Durán

Académico:

Samuel Báez Herrera

Experiencia Educativa: Desarrollo de sistemas en red

Fecha:

19/03/2020

Inicio

Como lenguaje se utilizó C en Windows con Mingw con g++ en su versión 8.1 y en Linux con GCC en su versión 9.3.

Como paso inicial, se comenzó creando un archivo de prueba, así como un pequeño programa en C el cual abriera y leyera el archivo. La siguiente implementación:

```
#include <stdio.h>

#include <stdlib.h>

main() {
    char ch;
    FILE *fp;
    printf("Abriendo archivo....\n");
    fp = fopen("test.txt", "r");
    printf("Archivo abierto, leyendo archivo....\n");
    if (fp == NULL){
        perror("Error abriendo el archivo, saliendo...\n");
        return 1;
    }
    printf("El archivo decia: \n");

    while((ch = fgetc(fp)) != EOF)
        printf("%c", ch);
    return 0;
}
```

Arrojo el siguiente resultado en la ejecución

```
Abriendo archivo....
Archivo abierto, leyendo archivo....
El archivo decia:
Este es un archivo de prueba
```

Descriptores de archivo

Posteriormente, se continuo a crear una solución para obtener el número máximo de descriptores de archivos, así como aumentar dicha cantidad. Por lo que se prosiguió a investigar la forma de realizarlo, llegando a la conclusión de que es imposible, al menos con C y MinGW, debido a que dicho compilador no funciona con POSIX y por esta razón no encuentra la librería "resource.h" por que busca ser nativo de Windows y quita dicha librería. Pero a grandes rasgos, el programa quedaría de la siguiente forma:

```
#include <stdio.h>
```

```
#include <sys/resource.h>
```

```
#include <string.h>
```

```
#include <errno.h>
```

```
#include <unistd.h>
```

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

```
#include <sys/stat.h>
```

```
#include <fcntl.h>
```

```
main() {
    char ch;
    FILE *fp;
    printf("Abriendo archivo....\n");
    fp = fopen("test.txt", "r");
    printf("Archivo abierto, leyendo archivo....\n");
    if (fp == NULL){
        perror("Error abriendo el archivo, saliendo...\n");
        return 1;
    }
    printf("El archivo decia: \n");

    while((ch = fgetc(fp)) != EOF)
        printf("%c", ch);

    if( getrlimit(RLIMIT_NOFILE, &old_lim) == 0)
        printf("Old limits -> soft limit= %ld \t"
```

```

        " hard limit= %ld \n", old_lim.rlim_cur,

        old_lim.rlim_max);

    else

        fprintf(stderr, "%s\n", strerror(errno));

    return 0;
}

```

Dicha solución, es parecida a la primera solución propuesta en el ejercicio anterior y no atribuyo a mi persona la creación de la parte resaltada.

Variables de entorno

A continuación, se busco la forma de obtener las variables de entorno, encontrando el siguiente fragmento:

```

printf("\n Obteniendo variables de entorno...\n");

const char* environment = getenv("PATH");

printf("PATH :%s\n", (environment!=NULL)? environment : "getenv returned NULL");

return 0;

```

Dando como resultado lo siguiente:

```

Obteniendo variables de entorno...
PATH :C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt v6-rev0\mingw32\opt\bin;C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt v6-rev0\mingw32\bin;C:\Program Files (x86)\mingw-w64\i686-8.1.0-posix-dwarf-rt v6-rev0\mingw32\bin;C:\Program Files\Intel\Intel(R) Management Engine Components\iCLS\;C:\WINDOWS\system32;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files\Intel\Intel(R) Management Engine Components\DAL;C:\Program Files (x86)\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\Intel\Intel(R) Management Engine Components\IPT;C:\Program Files\NVIDIA Corporation\NVIDIA NvDLISR;C:\Program Files (x86)\NVIDIA Corporation\PhysX\Common;C:\WINDOWS\system32;C:\WINDOWS\System32\Wbem;C:\WINDOWS\System32\WindowsPowerShell\v1.0\;C:\WINDOWS\System32\OpenSSH\;C:\Program Files\dotnet\;C:\Program Files\Microsoft SQL Server\Client SDK\ODBC\130\Tools\Binn\;C:\Program Files (x86)\GKSharp\2.12\bin;C:\sonar\sonar-scanner-msbuild-4.7.1.2311-net46;C:\OpenJDK\jdk-11\bin;C:\Program Files\Microsoft SQL Server\130\Tools\Binn\;C:\Program Files\Git\cmd;C:\apache-maven-3.6.3\bin;C:\apache-maven-3.6.3;C:\Program Files (x86)\BaseX\bin;C:\Program Files\MongoDB\Server\4.2\bin;C:\Users\Davis\AppData\Local\Microsoft\WindowsApps;C:\Users\Davis\AppData\Local\Programs\Microsoft VS Code\bin;C:\Users\Davis\.dotnet\tools;C:\Program Files\JetBrains\Intel liJ IDEA Educational Edition 2019.3.3\bin;

```

Nota: A partir de aquí, solo se pondrán los fragmentos agregados al código y al final se pondrá el código completo, para evitar repetición excesiva del código.

Manejo de procesos

Finalmente, la parte final de la practica indica algunas tareas con procesos, para la primer tarea es crearlo, por lo cual solo se debe agregar al programa la instrucción `exec()` para crear un proceso de la instancia en la que se encuentra:

```

printf("Creando un nuevo proceso desde el programa actual...\n");

char *args[]={"/Proceso",NULL};

execvp(args[0],args);

return 0;

```

Dando como resultado lo siguiente:

```
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
Creando un nuevo proceso desde el programa actual...
```

Esto debido a que el programa crea un proceso nuevo de si mismo y el siguiente crea otro.

Continuando, se debía bifurcar el proceso actual y obtener el pid tanto del padre como del hijo, por lo que se implemento de la siguiente manera:

```
printf("Bifurcando el proceso actual...\n");
int pidchild = fork();
if( 0 == pidchild)
{
    printf( "PID del hijo %ld\n", (long)getpid());
}
else
{
    printf( "PID del padre: %ld\n", (long)getpid());
}
return 0;
```

Obteniendo como resultado lo siguiente:

```
Bifurcando el proceso actual...
PID del padre: 10795
PID del hijo 10802
```

A continuación, se buscaba bloquear un proceso, esto se hizo bloqueando el proceso padre para que no saliera hasta que terminara el hijo, solo se agregaron unas modificaciones al código anterior:

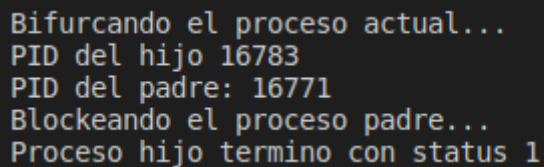
```

int status;

printf("Bifurcando el proceso actual...\n");
int pidchild = fork();
if( 0 == pidchild)
{
    printf( "PID del hijo %ld\n", (long)getpid());
    exit(1);
}
else
{
    printf( "PID del padre: %ld\n", (long)getpid());
    printf("Bloqueando el proceso padre... \n");
    if (wait(&status) >= 0)
    {
        printf("Proceso hijo termino con status %d \n", WEXITSTATUS(status));
    }
}
return 0;

```

Obteniendo el siguiente resultado:



```

Bifurcando el proceso actual...
PID del hijo 16783
PID del padre: 16771
Bloqueando el proceso padre...
Proceso hijo termino con status 1

```

Para terminar, se busco matar a un proceso, en este caso el hijo, por lo que, de la misma manera, se hicieron algunas modificaciones al mismo código:

```

int status;

printf("Bifurcando el proceso actual...\n");
int pidchild = fork();
if( 0 == pidchild)
{

```

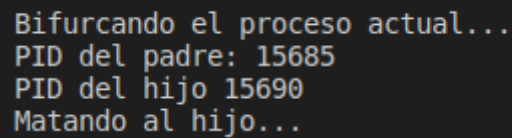
```

        printf( "PID del hijo %ld\n", (long)getpid());
        printf("Matando al hijo... \n");
        kill(pidchild, SIGKILL);
        printf("Hijo muerto... \n");
    }
    else
    {
        printf( "PID del padre: %ld\n", (long)getpid());
    }

    return 0;

```

Obteniendo como resultado:



```

Bifurcando el proceso actual...
PID del padre: 15685
PID del hijo 15690
Matando al hijo...

```

Como aclaración final, ciertas partes se realizaron en Linux, debido a su facilidad para crear procesos, por lo que el código final quedaría de la siguiente manera:

```

#include <stdio.h>

#include <string.h>

#include <windows.h>

#include <sys/wait.h>

#include <signal.h>

main() {
    char ch;
    FILE *fp;
    printf("Abriendo archivo....\n");
    fp = fopen("test.txt", "r");
    printf("Archivo abierto, leyendo archivo....\n");

```

```

if (fp == NULL){
    perror("Error abriendo el archivo, saliendo...\n");
    return 1;
}
printf("El archivo decia: \n");

while((ch = fgetc(fp)) != EOF)
    printf("%c", ch);

// if( getrlimit(RLIMIT_NOFILE, &old_lim) == 0)
//     printf("Old limits -> soft limit= %ld \t"
//           " hard limit= %ld \n", old_lim.rlim_cur,
//           old_lim.rlim_max);
// else
//     fprintf(stderr, "%s\n", strerror(errno));

printf("\nObteniendo variables de entorno...\n");
const char* environment = getenv("PATH");
printf("PATH :%s\n",(environment!=NULL)? environment : "getenv returned NULL");

    // printf("Creando un nuevo proceso desde el programa actual...\n");
// char *args[]={"/Proceso",NULL};
// execvp(args[0],args);
// return 0;
// int status;
// printf("Bifurcando el proceso actual...\n");
// int pidchild = fork();
// if( 0 == pidchild)
// {
//     printf( "PID del hijo %ld\n", (long)getpid());
//     exit(1);

```



```

// }
// else
// {
//     printf( "PID del padre: %ld\n", (long)getpid());
//     printf("Bloqueando el proceso padre... \n");
//     if (wait(&status) >= 0)
//     {
//         printf("Proceso hijo termino con status %d \n", WEXITSTATUS(status));
//     }
// }

// return 0;

    int status;
    printf("Bifurcando el proceso actual...\n");
    int pidchild = fork();
    if( 0 == pidchild)
    {
        printf( "PID del hijo %ld\n", (long)getpid());
        printf("Matando al hijo... \n");
        kill(pidchild, SIGKILL);
        printf("Hijo muerto... \n");
    }
    else
    {
        printf( "PID del padre: %ld\n", (long)getpid());
    }

    return 0;

}

```