

PRACTICA 1

Luis Guillermo Rivera Stephens 746651

Diego Rosales Rojas 744561

Emular el ambiente del sistema operativo que permite que los usuarios creen sesiones en el sistema y ejecuten procesos.

PROGRAMAS Y EXPLICACION

Init.c

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

void killsignal(){
    printf("Eliminando proceso padre");
    kill(0, SIGKILL);
}

int init(){
    signal(SIGTERM, killsignal);
    int pid = getpid();
    FILE *archivo = fopen("pid.txt", "w");

    if (archivo == NULL) {
        printf("Error al abrir el archivo");
        return 0;
    }
}
```

```

    fprintf(archivo, "%d\n", pid);
    fclose(archivo);

    for (int i = 0; i<6; i++) {
        if (fork() == 0) {
            execlp("xterm", "xterm", "-e", "./getty.exe", NUL
        }
    }
    while(1) {
        wait(NULL);
        if (fork() == 0) {
            execlp("xterm", "xterm", "-e", "./getty.exe", NUL
        }
    }
}

int main(){
    init();
}

```

1. El programa al iniciar guarda el PID del padre para poder enviar la señal desde un proceso hijo, no sin antes registrar el handler junto con la señal que va a manejar
2. Hace un for loop para crear los 6 procesos hijos, y cada que pase un proceso hijo lo vamos a reemplazar por la ejecución del programa getty
3. Después hacemos un loop infinito el cual va a esperar que alguno de los procesos hijos se cierre de manera inadecuada, por ejemplo presionando la cruz de la terminal o presionando `Ctrl C` , en cuanto detecte que un proceso se cerró abrirá otro idéntico

Getty.c

```

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

int validate_username(char *username, char *password){
    FILE *f=fopen("passwd.txt","r");
    char line[256];

    if(!f)
    {
        printf("Error al abrir el archivo passwd");
        return 0;
    }

    while (fgets(line, sizeof(line),f)!= NULL)
    {
        char *user =strtok(line,":");
        char *pass =strtok(NULL,"\n");

        if (strcmp(user,username)==0 && strcmp(pass,password)
        {
            fclose(f);
            return 1;
        }
    }
    fclose(f);
    return 0;
}

void getty(){
    char username[256];
    char password[256];

```

```

while(1){
    printf("Username: ");
    fgets(username, sizeof(username), stdin);
    username[strcspn(username, "\n")] = '\0';

    printf("Password: ");
    fgets(password, sizeof(username), stdin);
    password[strcspn(password, "\n")] = '\0';

    if (validate_username(username, password))
    {
        int pid=fork();

        if (pid==0){
            execlp("./sh.exe", "sh.exe", NULL);
        }
        else{
            wait(NULL);
            printf("Volviendo a iniciar cuenta\n");
        }
    }
    else{
        printf("Contraseña o usuario incorrecto, intente
    }

}

int main(){
    getty();
}

```

1. Primero declaramos la función para validar el usuario, la cual con un ciclo que revisa cada línea y la compara con el usuario y la contraseña brindada, obvio separando el usuario y la contraseña en dos apuntadores distintos
2. Se piden el usuario y la contraseña al usuario y se reemplaza el salto de línea con el finalizador de la cadena
3. Se llama a la función que creamos y si retorna 1 (True) entonces hace al hijo proceso y en el ejecuta el shell mientras que el padre espera

Sh.c

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

int main() {

    FILE *archivo = fopen("pid.txt", "r");
    if (archivo == NULL) {
        printf("Error al abrir el archivo");
        return -1;
    }

    int pid;
    if (fscanf(archivo, "%d", &pid) != 1) {
        printf("Error al leer el PID del archivo");
        fclose(archivo);
        return 0;
    }

    fclose(archivo);
```

```

char command[100];
char *args[20];

while (1) {
    printf("Programa a ejecutar: ");
    fgets(command, sizeof(command), stdin);

    command[strcspn(command, "\n")] = 0;

    if (strcmp(command, "exit") == 0) {
        printf("Saliendo...\n");
        break;
    }
    if (strcmp(command, "shutdown") == 0) {
        printf("Apagando...\n");
        kill(pid, SIGTERM);
    }

    int i = 0;
    args[i] = strtok(command, " ");
    while (args[i] != NULL && i < 9) {
        i++;
        args[i] = strtok(NULL, " ");
    }
    args[i] = NULL;

    int p = fork();
    if (p == 0) {
        execvp(args[0], args);

    } else {
        wait(NULL);
    }
}

```

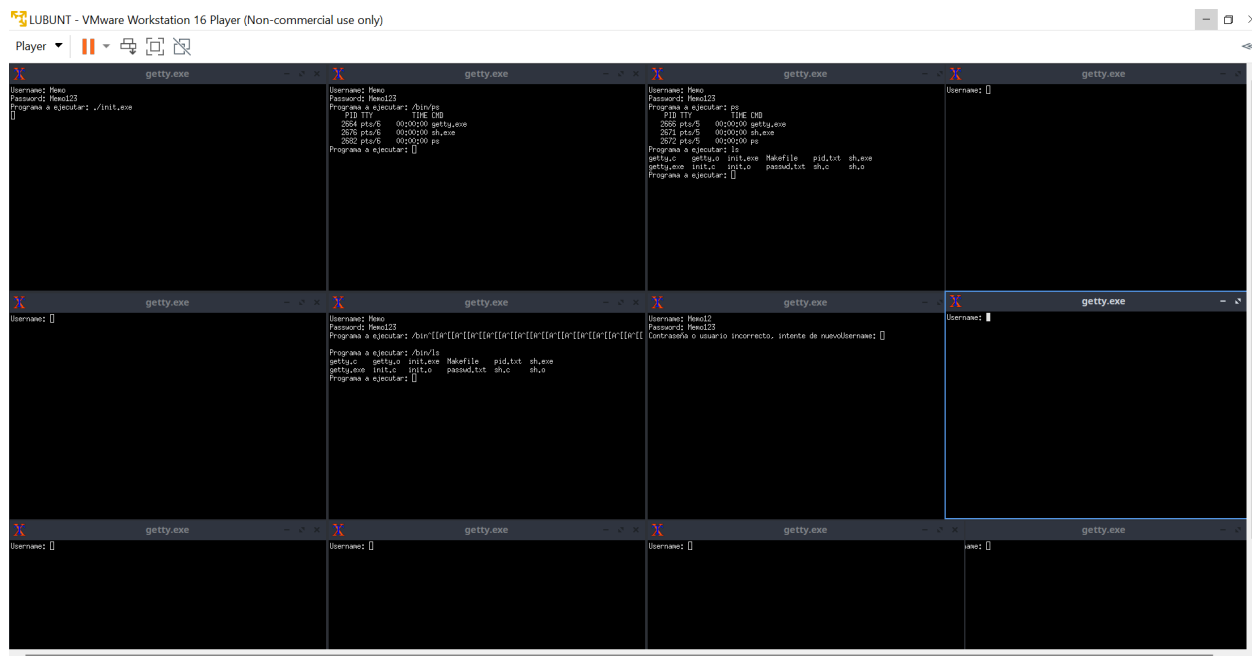
```

    return 0;
}

```

1. Primero guardamos en una variable el PID padre que guardamos en el archivo de texto
2. Se inicia un ciclo while para ir recibiendo los comandos a ejecutar en el shell
3. Si el comando es `exit` saldrá del ciclo y terminara el programa, solo para que se vuelva a repetir el getty
4. Si el comando es `shutdown` enviara una señal al proceso padre, misma señal la cual ya tiene su respectivo handler
5. En caso de no ser ninguno de los anteriores, se separa en argumentos y se ejecuta el proceso en un proceso hijo, el proceso padre espera a que termine y sigue ejecutando el shell

EJEMPLOS DE EJECUCIÓN



PREGUNTAS

Revise el comando ps, y qué opciones tiene para su ejecución, explique ¿por qué cuando un usuario teclea el comando ps sin parámetros solo muestra unos cuantos procesos?

Porque ps solo muestra los procesos que están vinculados a esa terminal

Al hacer el init solo mostrará los procesos

Para ver todos los procesos se puede usar ps -a

Al ejecutar el proceso init ¿Qué procesos nuevos se muestran en el sistema?


```
fso@fsovn:~/Desktop/FundamentosDeSistemasOperativos/practica1$ make &
[1] 2412
fso@fsovn:~/Desktop/FundamentosDeSistemasOperativos/practica1$ echo "Iniciando el proceso de compilacion"
Iniciando el proceso de compilacion
rm -f *.o *.exe
gcc -c init.c
gcc -o init.exe init.o
gcc -c sh.c
gcc -o sh.exe sh.o
gcc -c getty.c
gcc -o getty.exe getty.o
./init.exe
^C
fso@fsovn:~/Desktop/FundamentosDeSistemasOperativos/practica1$ ps
  PID TTY          TIME CMD
 1890 pts/0    00:00:00 bash
 2412 pts/0    00:00:00 make
 2434 pts/0    00:00:00 init.exe
 2435 pts/0    00:00:00 xterm
 2436 pts/0    00:00:00 xterm
 2437 pts/0    00:00:00 xterm
 2438 pts/0    00:00:00 xterm
 2439 pts/0    00:00:00 xterm
 2440 pts/0    00:00:00 xterm
 2447 pts/0    00:00:00 ps
fso@fsovn:~/Desktop/FundamentosDeSistemasOperativos/practica1$
```

Se muestran los procesos del Xterm (getty), ojo esto desde la misma terminal de init.exe, pero no se muestran los sh incluso si se están ejecutando

Inicie al menos dos sesiones en las ventanas que creó getty y muestre la lista de procesos en la misma ventana donde ejecutó el proceso init, ¿qué procesos nuevos se muestran en el sistema?'

No habría procesos nuevos mostrados, pues ese proceso sería ejecutado en otra terminal (en las de xterm), a menos de que lo ejecutemos con el -a


```
fso@fsovm:~/Desktop/FundamentosDeSistemasOperativos/practica1$ make &
[1] 2515
fso@fsovm:~/Desktop/FundamentosDeSistemasOperativos/practica1$ echo "Iniciando el proceso de compilacion"
Iniciando el proceso de compilacion
rm -f *.o *.exe
gcc -c init.c
gcc -o init.exe init.o
gcc -c sh.c
gcc -o sh.exe sh.o
gcc -c getty.c
gcc -o getty.exe getty.o
./init.exe
^C
fso@fsovm:~/Desktop/FundamentosDeSistemasOperativos/practica1$ ps
  PID TTY          TIME CMD
 1890 pts/0    00:00:00 bash
 2515 pts/0    00:00:00 make
 2537 pts/0    00:00:00 init.exe
 2538 pts/0    00:00:00 xterm
 2539 pts/0    00:00:00 xterm
 2540 pts/0    00:00:00 xterm
 2541 pts/0    00:00:00 xterm
 2542 pts/0    00:00:00 xterm
 2543 pts/0    00:00:00 xterm
 2554 pts/0    00:00:00 ps
fso@fsovm:~/Desktop/FundamentosDeSistemasOperativos/practica1$ ps -a
  PID TTY          TIME CMD
 2515 pts/0    00:00:00 make
 2537 pts/0    00:00:00 init.exe
 2538 pts/0    00:00:00 xterm
 2539 pts/0    00:00:00 xterm
 2540 pts/0    00:00:00 xterm
 2541 pts/0    00:00:00 xterm
 2542 pts/0    00:00:00 xterm
 2543 pts/0    00:00:00 xterm
 2552 pts/1    00:00:00 sh.exe
 2555 pts/0    00:00:00 ps
fso@fsovm:~/Desktop/FundamentosDeSistemasOperativos/practica1$
```

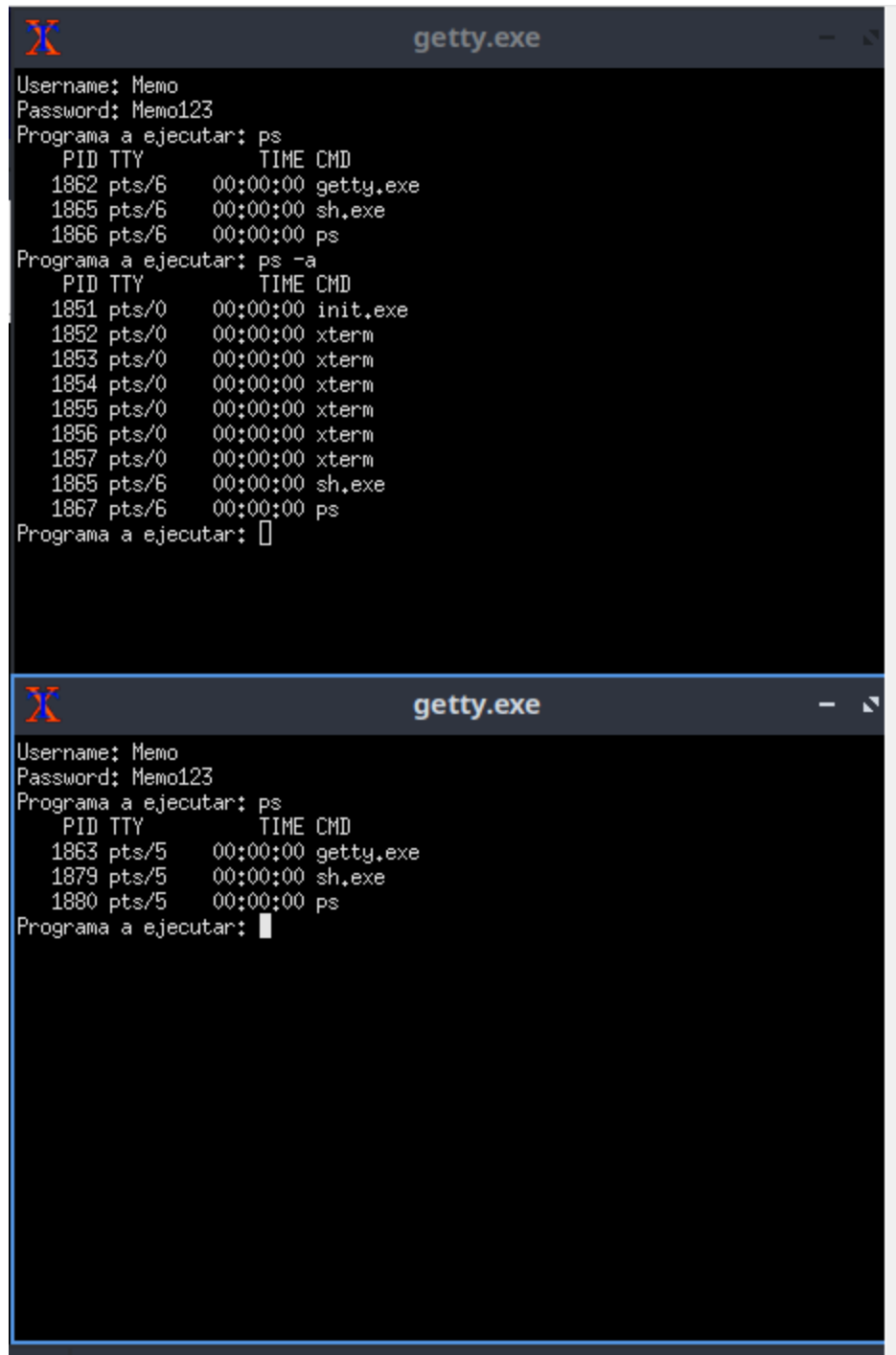


En una de las ventanas del shell creada por el proceso getty, tecleé el comando ps, ¿qué procesos se muestran?

Si solo ejecutamos el comando ps nos da los un getty.exe y un sh.exe, ojo solo ingresando a un shell, por lo mismo que ps muestra los procesos referentes a esa terminal

Como getty ejecuta una terminal no muestra el init.exe

Antes de eliminar los procesos con el shutdown se muestra el proceso init.exe, 6 procesos de xterm(getty) y depende de a cuantos shells hayas iniciado se verán reflejados en el ps -a



```
Username: Memo
Password: Memo123
Programa a ejecutar: ps
  PID TTY          TIME CMD
  1862 pts/6        00:00:00 getty.exe
  1865 pts/6        00:00:00 sh.exe
  1866 pts/6        00:00:00 ps
Programa a ejecutar: ps -a
  PID TTY          TIME CMD
  1851 pts/0        00:00:00 init.exe
  1852 pts/0        00:00:00 xterm
  1853 pts/0        00:00:00 xterm
  1854 pts/0        00:00:00 xterm
  1855 pts/0        00:00:00 xterm
  1856 pts/0        00:00:00 xterm
  1857 pts/0        00:00:00 xterm
  1865 pts/6        00:00:00 sh.exe
  1867 pts/6        00:00:00 ps
Programa a ejecutar: 
```

```
Username: Memo
Password: Memo123
Programa a ejecutar: ps
  PID TTY          TIME CMD
  1863 pts/5        00:00:00 getty.exe
  1879 pts/5        00:00:00 sh.exe
  1880 pts/5        00:00:00 ps
Programa a ejecutar: 
```

¿Qué efecto tiene la espera de un proceso hijo en el proceso getty?, ¿qué sucedería si no existiera esa espera?

Los procesos sh quedarían como procesos zombies y si el padre no recolecta el fin de cada proceso, quedarían almacenados en la tabla de procesos. Aparte de que como es un loop infinito no dejaría poner el usuario ni siquiera.

En cambio si se recolectan estos, daría fin a su ejecución y terminaría con el loop, además de poder acabar con el resto de procesos.

En los procesos anteriores está utilizando la llamada exec que para reemplazar la imagen de un proceso busca el programa ejecutable en los directorios especificados en la variable de ambiente PATH. Investigue (buscar en documentación de UNIX) por qué esta llamada pueda hacer uso de los valores en la variable PATH sin que sea necesario inicializar esta variable en cada uno de sus procesos.

Porque los procesos hijos heredan el entorno del padre por esta razón, al solo definir en el padre no es necesario redefinir en los demás procesos ya que tienen o están en el mismo entorno que este. Esto quiere decir que cuando se crea un proceso con fork() la variable PATH ya está inicializada por medio del padre y puede buscar los directorios de los ejecutables. Las llamadas exec funcionan de manera esperada sin necesidad de redefinir PATH en cada proceso nuevo.

Apuntes.de. (s.f.). *La variable de entorno PATH*. Recuperado de <https://apuntes.de/linux-certificacion-lpi/la-variable-de-entorno-path/>.