I OBJETIVOS

Al término de esta práctica el alumno debe ser capaz de:

- Usar los servicios que provee el sistema operativo para crear procesos, reemplazar la imagen de un proceso y esperar procesos
- Simular un ambiente donde el sistema operativo crea nuevos procesos para que los usuarios puedan crear sesiones y como se mantienen las preferencias de la sesión.
- Diseñar y desarrollar shell en modo texto.

II BIBLIOGRAFÍA

- Silberschatz, Galvin, Gagne, "SISTEMAS OPERATIVOS", McGraw Hill, 7ª Ed.
- William Stallings, "SISTEMAS OPERATIVOS",Prentice Hall, 4ª Ed.
- Neil Matthew & Richard Stones, "BEGINNING LINUX PROGRAMMING", Wrox, 2ª Ed.

III RECURSOS

- Una estación de trabajo con Linux con su ambiente gráfico.
- Un editor de texto.
- El compilador de C de UNIX.

IV ACTIVIDADES

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

1.1 Programas a desarrollar

Para esta emulación será necesario desarrollar 3 programas, que serán ejecutados como los procesos:

- `init`
- `getty`
- `sh`

1.2 El proceso `init`

El proceso llamado `init` creará 6 procesos hijos cuya imagen será reemplazada por procesos `getty`. Para que la ejecución de los procesos `getty` creados en esta práctica sean claramente visibles, cada uno de ellos serán creados a través de una ventana `xterm`, vea como ejecutar procesos a través de una ventana nueva de `xterm`.

El programa `xterm` abre una ventana nueva en el ambiente gráfico y con el parámetro `-e` permite la ejecución de un nuevo proceso en esa ventana. Ejemplo:

```
$ xterm -e ./programa
```

Para mayor información sobre `xterm`, consulte la ayuda en línea con el comando `man`.

Una vez que el proceso `init` deje en ejecución los 6 procesos `getty`, este estará verificando que esos 6 procesos estén siempre en ejecución, si uno de estos es terminado porque el usuario cierra la ventana de `xterm`, entonces `init` deberá crear uno nuevo. La única forma de hacer que `init` termine será con la ejecución del comando `shutdown` que podrá ser introducido desde el shell.

1.3 Los procesos `getty`

Los procesos `getty` mostrarán al usuario el prompt de login y password para iniciar una sesión, tal como ocurre cuando se inicia una sesión de UNIX desde una terminal en modo texto. El login y password será validado de un archivo de texto llamado `shadow`, la estructura de este archivo es como la que se muestra en la Figura 1. Una vez que el el usuario se valida, `getty` **creará un proceso hijo** que será reemplazado por una sesión del shell (proceso `sh`).

Una vez que `getty` inicia la sesión del shell, este esperará a que el proceso `sh` termine, de manera que cuando el usuario salga del shell con un comando `exit`, el proceso `getty` volverá a pedir el prompt de login y password al usuario para que posteriormente vuelva a crear la función de `sh`.

```
Pepito:Mipassword
Maria:abc0123
…
```

Figura 1. Ejemplo del archivo de passwords.

1.4 El proceso `sh`

Se desarrollará un programa `sh` que hará la función de un intérprete de comandos. Al ejecutarse, el proceso `sh` estará pidiendo cadenas de caracteres al usuario, estas cadenas de caracteres pueden ser nombres de programas que serán ejecutados como procesos o cualquiera de los comandos `exit` o `shutdown`.

El comando `exit` del shell será para terminar la ejecución de `sh` permitiendo que el proceso `getty` retome el control pidiendo nuevamente un nombre de usuario y un password.

El comando `shutdown` del shell terminará la ejecución de todos los procesos del sistema, estos son todos los procesos `shell`, `getty` y `init` que estén en ejecución.

1.4.1 Variables de ambiente

Los comandos para manejo de variables de ambiente que serán soportados por `shell` son `export` y `echo`.

El comando `export` nos permitirá establecer el valor de una variable de ambiente, por ejemplo:

```
sh > export VARIABLE=VALOR
```

El comando `echo` mostrará el valor de una variable de ambiente en pantalla
, por ejemplo:

```
sh > echo $VARIABLE
```

Una variable de ambiente podrá sustituir **cualquier parte**[1] de una línea de comando, incluyendo el comando como se muestra en el siguiente ejemplo de ejecución de procesos en la parte 1.4.2.

```
sh > export COMANDO=/bin/ls
sh > $COMANDO
```

La variable de ambiente `path`

Una de las variables de ambiente que podrá ser agregarse en el shell es la variable de ambiente `PATH`, esta variable de ambiente indica un directorio donde serán buscados los programas ejecutables que no son encontrados en el directorio actual. En el siguiente ejemplo se muestra como establecemos la variable de ambiente `PATH` con el valor `/bin` y al ejecutar el programa `ls` y no ser encontrado en el directorio actual, este es buscado como `/bin/ls`

```
sh > export PATH=/bin
```

[1] Si la sustitución solo funciona en partes restringidas de una línea de comandos, se reducirán puntos de la calificación.

```
sh > ls
```

1.4.2 Ejecución de otros procesos

El shell permitirá lanzar la ejecución de nuevos procesos a partir de cualquier programa existente. Esta ejecución puede realizarse en primer plano o en segundo plano. Cuando se ejecuta un proceso en primer plano, el shell se bloquea esperando a que el proceso termine su ejecución, en cambio, cuando la ejecución es en segundo plano, el shell continúa en ejecución sin esperar a que el proceso nuevo termine.

Ejemplos:

Ejemplo 1:
Muestra los archivos que existen en el directorio actual.

```
sh > /bin/ls
```

Ejemplo 2:
Ejecuta el programa `ls` que se encuentra en el directorio indicado en la variable de ambiente `PATH`

```
sh > ls
```

Ejemplo 3:
Muestra los procesos en ejecución.

```
sh > /bin/ps
```

Ejemplo 4:
Ejecuta el programa llamado `programax` que se encuentra en el directorio actual.

```
sh > ./programax
```

Ejemplo 5:
Ejecuta **en segundo plano** el programa llamado `programax` que se encuentra en el directorio actual.

```
sh > ./programax &
```

1.5 Restricciones y recomendaciones

En ninguno de los ejercicios anteriores podrá utilizarse la llamada `system()`, utilice las llamadas `fork()` y `exec`[2] para la creación y ejecución de nuevos procesos.

2 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?

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

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

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

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

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 la variable en cada uno de sus procesos.

[2] `exec`: se refiere a cualquiera de las variantes de la llamada `exec`, estas pueden ser: `excel()`, `execlp()`, `execle()`, `execv()`, `execvp()` o `execve()`, la llamada recomendada para esta parte es `execvp()`.

V ENTREGA

No incluya líneas de código en sus programas de las cuales desconozca su funcionamiento. El código no conocido será anulado en el funcionamiento de la práctica.

1 Entrega y Revisión

Entregar en el apartado correspondiente de Moodle un archivo ZIP que contenga:

- Los programas fuentes que se piden: `init.c`, `getty.c` y `sh.c`.
- El archivo `Makefile` con el que se puedan compilar todos los programas tecleando solamente el comando `make`.

El tiempo límite para entregar miércoles 20 de Febrero a las 11:59 PM. La revisión se realizará después de la semana 7.

2 Equipos

Esta práctica se hará en equipos (máximo 3 integrantes), es necesario que en la revisión esté el equipo completo ya que el integrante que no se presente no tendrá calificación en la práctica.

Importante: Al indicarse que el trabajo debe ser desarrollado por equipos, se entiende que no se permite colaboración entre equipos, cualquier evidencia de esto será considerada plagio.

3 Evaluación

Puntualidad en la entrega	El producto fue entregado a tiempo y de acuerdo a las especificaciones/formato de entrega. Si son dos o más sesiones de revisión muestra los avances solicitados en todas las sesiones		El producto que se entregó a tiempo tiene alguna falla y solicitan que se les revise una versión más nueva o no fue entregado de acuerdo a las especificaciones/formato de entrega o en alguna de las sesiones de revisión no tenían los avances esperados.
	+10		0
Puntualidad en las revisiones	El equipo estuvo completo y puntual en todas las sesiones de revisión.	Si hubo dos o más sesiones con el equipo, el equipo estuvo completo y puntual en casi todas las sesiones de revisión	Si solo hubo una sesión de revisión, el equipo no estuvo completo o no fue puntual. Si fueron dos o más sesiones de revisión, en más de una sesión el equipo no estuvo completo o fue puntual
	+5	+2.5	0
Funcionamiento	El producto cumple con todas las especificaciones indicadas en el documento y no tiene fallas	El producto muestra una falla no esperada o el producto está casi completo, puede funcionar excepto la parte no completada	El producto muestra más de una falla inesperada o no funciona, esto puede ser debido a que no esté completo.
	+25	+12.5	0
Interfaz con el usuario	El producto funciona y pudo ser utilizado sin necesidad de recibir indicaciones por el desarrollador, tiene instrucciones claras para ser utilizado.	El producto funciona, pero hubo necesidad de recibir alguna indicación para su uso por parte del desarrollador del producto	El producto carece de instrucciones claras para ser utilizado y requiere que alguno de los desarrolladores esté presente para su utilización o no puede utilizarse debido a que no está completo
	+5	+2.5	0
Claridad en el código	El código es claro, usa nombres de variables adecuadas, está debidamente comentado e indentado. Puede ser entendido por cualquier otra persona que no intervino en su desarrollo.	El código carece de claridad, puede ser entendido por cualquier persona ajena a su desarrollo pero con cierta dificultad.	El código carece de comentarios, está mal indentado, usa nombres de variables no adecuadas.
	+5	+2.5	0
Defensa del producto	Todos los integrantes son capaces de explicar cualquier parte del producto presentado	Alguno de los integrantes muestra dudas sobre alguna parte del desarrollo del producto presentado	Más de un integrante, o si el trabajo fue individual , el desarrollador duda sobre cómo está desarrollado producto
	+50	+25	0
Sobresaliente 20 %	Tiene 1 en todos los puntos anteriores. El producto entregado es sobresaliente, muestra tener la calidad para ser expuesto como un producto representativo de la carrera Hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado		No tiene 1 en todos los puntos anteriores, o el producto entregado no es sobresaliente y no muestra tener la calidad para ser expuesto como un producto representativo de la carrera o no hay evidencia de que los desarrolladores se documentaron y muestran aprendizajes más allá de lo esperado
	+20		0