SISTEMAS OPERATIVOS II

Tercer curso Ingeniería Técnica Informática de sistemas. Curso 2006-2007

Práctica 3: Procesos en UNIX: Entorno, prioridades

Continuar la codificación de un intérprete de comandos (shell) en UNIX. Al igual que en la práctica anterior

- los argumentos entre corchetes [] son opcionales.
- Los argumentos separados por `|` indican que debe ir uno u otro, pero no ambos simultaneamente.
- Cuando el shell no pueda ejecutar una acción por algún motivo, debe indicarlo con un mensaje como el que se obtiene con `sys_errlist[errno]` o con *perror()*
- El shell no debe dilapidar memoria. La memoria que se asigna (con *malloc()*) y deja de utilizarse debe ser liberada.
- La información que se nuestra en pantalla no debe incluir en ningun caso líneas en blanco.
- El shell leerá de su entrada estándar y escribirá en su salida estándar, de manera que podría ser ejecutado un archivo de comandos invocando el shell con su entrada estándar redireccionada a dicho archivo.

En esta práctica el shell llevará cuenta de los procesos que tiene en segundo plano, informando de su estado con el comando *jobs* que se cita a continuación. La implementación de la lista de procesos en segundo plano es libre y solo debe llevarse contabilidad de los procesos en segundo plano lanzados desde este shell con el comando *background*; consecuentemente deberá modificarse el comando *fork*. Además se permitirá la creación de procesos y ejecución de programas con el entorno y/o las prioridades modificadas. El manejo de las prioridades se hará con las llamadas *getpriority* y *setpriority*

getenv [-e|-a] Muestra la lista de todas las variables de entorno. Para cada variable de entorno muestra, en una misma línea, la variable (nombre=valor), la dirección de memoria donde se almacena la variable (el valor de la variable e[i] como puntero) y la dirección de memoria donde se almacena el puntero (&e[i]). El acceso será mediante *environ* (-e) o mediante el tercer argumento de main (-a).

entornos Muestra el valor (como punteros) del tercer argumento de main y de *environ* así como las direcciones de memoria donde están almacenados el tercer argumento de main y environ. Ejemplo

```
#getenv
......
.......
.......
ffbefcb8->env[29]=(ffbeff86) PATH=/usr/bin:/usr/local/bin:
ffbefcbc->env[30]=(ffbeffa5) LC_MONETARY=es
ffbefcc0->env[31]=(ffbeffb4) LC_COLLATE=es
ffbefcc4->env[32]=(ffbeffc2) _INIT_NET_STRATEGY=none
#entornos
20a34->environ=ffbefc44 | ffbefc24->env=ffbefc44  | ffbefc20->argv=ffbefc3c
#
```

get var1, var2 ... Devuelve los valores de las variables de entorno var1, var2 .. obtenidos mediante el tercer argumento de *main*, *environ* y la función de librería *getenv*. Para cada caso imprime además la dirección de memoria donde está la variable (su valor como puntero) y la dirección donde se almacena el puntero (salvo en el caso de obtenerla con *getenv*).

put -e|-a|-p var valor Establece el valor de la variable de entorno *var* a *valor* accediendo a ella mediante *environ* (-e), el tercer argumento de main (-a) o la función de librería *putenv*. Si la variable no existe no la creará (informará de ello), salvo que se haya especificado que se haga mediante *puntenv*, entonces será creada.

sus -e|-a var nue=val Sustituye de la variable de entorno *var* por *nue=val* accediendo a ella mediante *environ* (-e) o el tercer argumento de main (-a). Si la variable no existe informará de ello.

getpri [pid] Informa de la prioridad del proceso *pid*. Si no se suministra *pid* informa de la prioridad del shell

setpri [pid] pri Establece la prioridad del proceso *pid* a pri. Si no se suministra *pid* establece la prioridad del shell

- modificar los comandos **background, exec y *comando*** para que pueda especificarse la prioridad y un entorno alternativo. El formato es
  [background|exec] [lista_variables] ej [args] [*pri]
  donde
  - Si no se especifica background (ejecución en segundo plano) o exec (reemplazo del código del shell) en el primer término, se

entenderá que la ejecución es en primer plano

- Si no se especifica lista_variables se entenderá que la ejecución es mediante *execv*. Lista variables puede contener solamente nombres de variables (en este caso su valor se obtiene de *environ*) o cadenas de la forma variable=valor (en este caso la variable no tiene por que existir previamente). El programa se ejecutará con un entorno que contiene UNICAMENTE las variables de la lista, usando para ello *execve*. Si como lista indicamos ENTORNONULO el programa se ejecutará con un entorno vacio.
- Como en la práctica anterior, al ejecutable podrá pasársele un número variable de argumentos y ademaás debe residir en uno de los directorios de la ruta de búsqueda o especificarse la trayectoria hasta él comenzando por /, ./ o ../
- El último término indica el nuevo valor de la prioridad con la que se ejecutará el programa. Este término tambien es opcional.

ejemplos

```
# background xterm -e ksh *12

# exec TERM DISPLAY HOME HZ=120 N=65 ./a.out arg1 arg2 *-3

# TERM DISPLAY HOME USER NUEVA=PRUEBA /usr/bin/xterm -hold -e ./a.out

# ENTORNONULO ./a.out
```

jobs[**-t**] [**-s**] [**-p**] [**-a**] Muestra una lista de los procesos ejecutados en segundo plano por ese shell. Para cada proceso mostrar, EN UNA SOLA LINEA, el pid, el momento en que fue lanzado, información del estado actual (activo, parado o terminado, junto con valor devuelto o señal según corresponda), la línea de comando con que fue lanzado y la prioridad. La opción -t no mostrará solo los que hayan terminado normalmente, la opción -s los que hayan terminado debido a una señal, la opción -p los parados y la opción -a los activos

clearjobs[**-t|-s**] Con la opción -t eliminará de la lista los que hayan terminado normalmente; con -s los que hayan terminado debido a una señal. Si no se especifica opción eliminará todos los que hayan terminado.

proc *pid* Nos da la informacion ampliada del proceso *pid* que devuelve *wait4* en la estrcutura *rusage*

```
struct rusage {
   struct timeval ru_utime; /* user time used */
   struct timeval ru_stime; /* system time used */
   long   ru_maxrss;        /* maximum resident set size */
   long   ru_ixrss;         /* integral shared memory size */
   long   ru_idrss;         /* integral unshared data size */
   long   ru_isrss;         /* integral unshared stack size */
   long   ru_minflt;        /* page reclaims */
   long   ru_majflt;        /* page faults */
   long   ru_nswap;         /* swaps */
   long   ru_inblock;       /* block input operations */
   long   ru_oublock;       /* block output operations */
   long   ru_msgsnd;        /* messages sent */
   long   ru_msgrcv;        /* messages received */
   long   ru_nsignals;      /* signals received */
   long   ru_nvcsw;         /* voluntary context switches */
   long   ru_nivcsw;        /* involuntary context switches */
};
```

Información detallada de las llamadas al sistema y las funciones de la librería debe obternerse con man (execve, wait4, getenv, putenv..)

FORMA DE ENTREGA Como en prácticas anteriores

FECHA DE ENTREGA Viernes 12 Enero 2007