

# Práctica 3: Programas de usuario y multiprogramación

2019 – Sistemas Operativos 2

Licenciatura en ciencias de la computación

*Entrega: miércoles 8 de mayo*

**Nota:** debe utilizar el Subversion de la materia creando un subdirectorío por alumno/grupo en:

<https://svn.dcc.fceia.unr.edu.ar/svn-no-anon/lcc/R-412/Alumnos/2019/>

## 1. Introducción

La segunda fase de Nachos es el subsistema de multiprogramación. Como en la primera parte, ya se provee parte del código necesario. Se debe ampliarlo y mejorarlo.

Puede ser útil cambiar algún parámetro de la máquina para desarrollar mejores casos de prueba (por ejemplo la cantidad de memoria física). Sin embargo conviene tener siempre presente que el código dentro del directorio `machine` representa la máquina física sobre la que se ejecuta el SO, por eso en general **no se debe modificar el código dentro de ese directorio**.

## 2. Ejercicios

1. Al ejecutar programas de usuario, tendremos **dos espacios de direcciones**. El primero es el que utiliza el kernel de Nachos, el cual corre en la máquina x86. El segundo es el del proceso de usuario que corre **sobre la máquina MIPS simulada**, por lo tanto serán direcciones en la máquina simulada. Luego los punteros (arreglos y cadenas) no pueden ser intercambiados entre estos dos espacios de direcciones.

Provea una forma de copiar datos desde el núcleo al espacio de memoria virtual del usuario y viceversa. Debe tener también dos tipos de funciones, una que lea/escriba cadenas terminadas por cero y otra que lea  $N$  bytes. Estas funciones son:

```
void ReadBufferFromUser(int userAddress, char *outBuffer,
                        unsigned byteCount);
bool ReadStringFromUser(int userAddress, char *outString,
                        unsigned maxByteCount);
void WriteBufferToUser(const char *buffer, int userAddress,
                       unsigned byteCount);
void WriteStringToUser(const char *string, int userAddress);
```

**Sugerencia:** para acceder a la memoria del usuario puede usar los métodos `Machine::ReadMem` y `Machine::WriteMem`.

2. Implemente las llamadas al sistema y la administración de interrupciones. Se deben soportar todas las llamadas al sistema definidas en `syscall.h` exceptuando `Fork` y `Yield`. Para poder probar `Exec` y `Join` se debe realizar primero el punto 3. Se sugiere implementarlas en el siguiente orden: `Create`, `Read/Write` (a la consola inicialmente), `Open`, `Close`, `Exit`, `Join`, `Exec`.

Note que la implementación debe ser “a prueba de balas”, o sea que un programa de usuario no debe poder hacer nada que haga caer el sistema operativo (con la excepción de llamar explícitamente a `Halt`).

Para la implementación de las llamadas que acceden a la consola puede ser útil implementar una clase `SynchConsole`, que provea la abstracción de acceso sincronizado a la consola. En `userprog/prog_test.cc` está el comienzo de la implementación de `SynchConsole`. La clase de acceso sincronizado a disco (`SynchDisk`) puede servir de modelo. Tenga en cuenta que a diferencia del disco, en este caso un hilo queriendo escribir no debería bloquear a un hilo queriendo leer.

3. Implemente multiprogramación con rebanadas de tiempo (“time slicing”). Será necesario:
  - a) Proponer una manera de ubicar los marcos de la memoria física para que se puedan cargar múltiples programas en la memoria (ver `lib/bitmap.hh`).
  - b) Forzar cambios de contexto después de cierto número de tics del reloj. Note que, ahora que está definido `USERPROG`, `Scheduler` almacena y recupera el estado de la máquina en los cambios de contexto.

4. La llamada `Exec` no provee forma de pasar parámetros o argumentos al nuevo espacio de direcciones. Unix permite esto, por ejemplo, para pasar argumentos de línea de comando al nuevo espacio de direcciones. Implemente esta característica.

Hay dos formas de hacerlo, puede elegir cualquiera de las dos:

- a) Una es al estilo de Unix: copiar los argumentos en el fondo del espacio de direcciones virtuales de usuario (la pila) y pasar un puntero a los argumentos como parámetro a `main`, usando el registro `4` para pasar la cantidad y `5` para pasar el puntero.

Los registros `4` y `5` (también llamados `a0` y `a1`) se usan para el primer y segundo parámetros de función, de acuerdo a la convención de llamada de MIPS que usamos. Es decir, son el `argc` y el `argv` del `main`, respectivamente.

El siguiente esquema ilustra la organización de los datos en la pila:

**Sugerencia:** puede aprovechar las funciones que provee Nachos en el archivo `userprog/args.cc`.

- b) La otra forma es agregar una nueva llamada al sistema, que cada espacio de direcciones llame como primera cosa en `main` y obtenga los argumentos para el nuevo programa.

**Sugerencia:** para esta forma también puede aprovechar `userprog/args.hh`, pero deberá hacer algunas modificaciones a la función `WriteArgs`.

5. Programas de usuario:

- a) Escriba al menos dos programas de usuario utilitarios (como `cat` y `cp`). Guarde los archivos en el directorio `userland` como los que ya vienen con Nachos; tenga en cuenta que deberá modificar `userland/Makefile` para que se compilen al ejecutar `make`.

**Sugerencia:** puede tomar como referencia el archivo `userland/tiny_shell.c` provisto por Nachos.

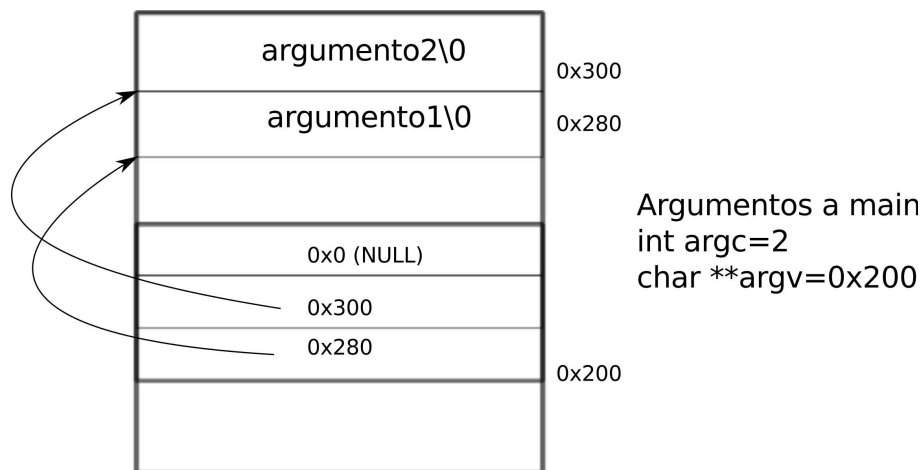


Figura 1: Esquema de la pila al pasar argumentos a `main`.

- b) Nachos incluye dos intérpretes de comandos: `userland/shell.c` y `userland/tiny_shell.c`.

Ambos leen líneas de comando de la consola y las ejecutan usando la llamada a sistema `Exec`. El primero es más avanzado, fundamentalmente en el manejo de argumentos.

Ambos programas tienen el mismo comportamiento al ejecutar una línea de comando: se quedan esperando a que termine la ejecución de esta para permitir ingresar un nuevo comando. Este comportamiento se llama ejecución en primer plano. Uno podría en cambio desear ejecutar tareas en segundo plano: que su ejecución no bloquee al intérprete, sino que este siga aceptando entradas y ejecutando otras tareas en forma paralela a la ejecución de la primera.

Modifique `shell` para que ejecute en segundo plano aquellas líneas de comando que empiecen con el carácter `&`.