# Plancha 3: Programas de usuario y multiprogramación

2024 – Sistemas Operativos 2

Licenciatura en Ciencias de la Computación

1. Al ejecutar programas de usuario, tendremos **dos espacios de direcciones**. El primero es el que utiliza el núcleo 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. Los punteros (arreglos y cadenas) no pueden ser intercambiados entre estos dos espacios de direcciones directamente.

   La cabecera `userprog/transfer.hh` ofrece funciones para copiar datos desde el núcleo al espacio de memoria virtual del usuario y viceversa. Las hay de dos tipos, unas que leen/escriben cadenas terminadas por ceros y otras que leen un número fijo de bytes. Las signaturas 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);
   ```

   Solo `ReadStringFromUser` está implementada. Agregue implementaciones para las demás.

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

2. Implemente las llamadas al sistema relacionadas con archivos (ver `syscall.h`): `Create`, `Remove`, `Open`, `Close`, `Read`, `Write`.

   Para implementar `Read` y `Write` a la consola (archivos 0 y1 respectivamente), crear una clase `SynchConsole` que provea la abstracción de acceso sincronizado a la consola. En `userprog/prog_test.cc` hay una función que sirve como comienzo de la implementación. La clase de acceso sincronizado a disco (`SynchDisk`) puede servir de modelo. Tener en cuenta que a diferencia del disco, en este caso un hilo queriendo escribir no debería bloquear a un hilo queriendo leer.

3. Implemente las llamadas a sistema `Exit`, `Exec` y `Join`.

   Para implementar multiprogramación deberá proponer una manera de ubicar los marcos de memoria física para que se puedan cargar múltiples programas en la memoria (ver `lib/bitmap.hh`). Recuerde además marcar la memoria como libre cada vez que finalice un programa.

   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 comparten el mismo comportamiento al ejecutar una línea de comando: permanecen en espera hasta que la ejecución de ésta finaliza para permitir el ingreso de un nuevo comando. Este comportamiento se llama ejecución en primer plano. Para probar la multitarea se deben 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.

Modifíquelos para que ejecuten en segundo plano aquellas líneas de comando que empiecen con el caracter `&`.

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 definiendo una nueva syscall (`Exec2`).

   Utilice una implementación similar a la 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 esquema de la Figura 1 ilustra la organización de los datos en la pila.

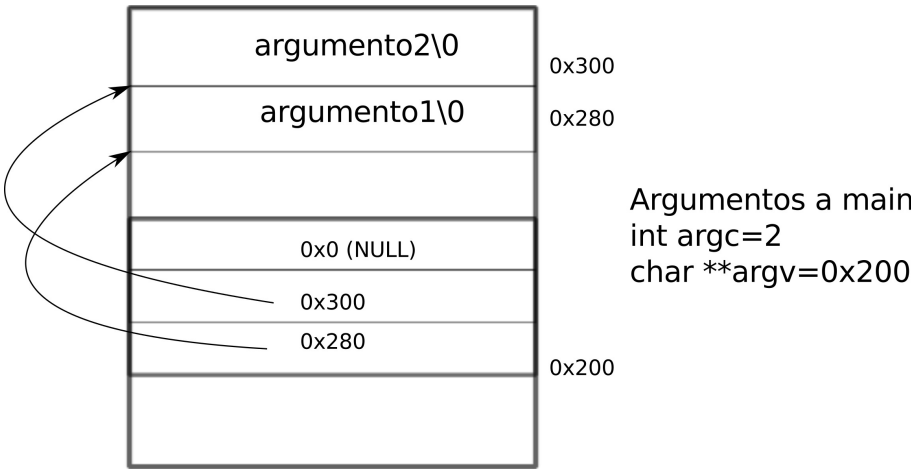


Figura 1: Esquema de la pila al pasar argumentos a `main`.

   **Sugerencia**: puede aprovechar las funciones que provee Nachos en el archivo `userprog/args.cc`. Además `userland/shell.c` muestra cómo llamar a `Exec2`[1] desde un programa de usuario.

5. Programas de usuario:

   *a*) Implemente una biblioteca de funciones para programas de usuario, en `userland/lib.c`. Debe incluir al menos las siguientes funciones:

   ```
   unsigned strlen(const char *s);
   void puts(const char *s);
   void itoa(int n, char *str);
   ```

   *b*) Escriba los programas utilitarios `cat`, `cp` y `rm`. Puede aprovechar la biblioteca del punto anterior para facilitar la implementación. 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.

[1] La llamada a `Exec2` está comentada y omite el sufijo 2 (agregarlo).