UT I Introducción a los sistemas operativos en tiempo real

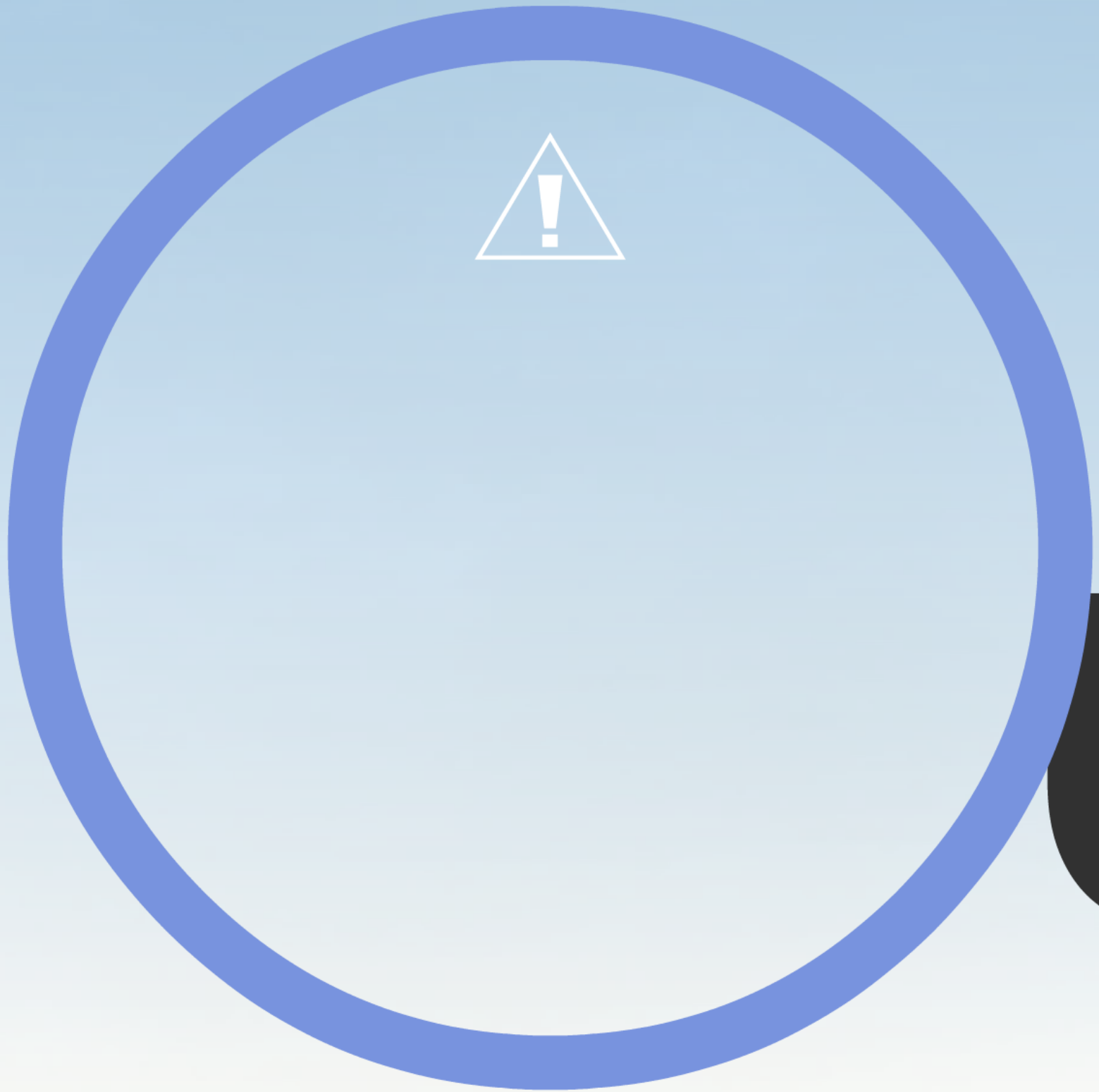
Analiza un sistema operativo en tiempo real en base a los sistemas operativos existentes

Prezi

UT I Introducción a los sistemas operativos en tiempo real

Analiza un sistema operativo en tiempo real en base a los sistemas operativos existentes

Prezi



U

Prezi

Booteo de sistemas operativos

La palabra booting es un apócope de la palabra bootstrap.

Se refiere al proceso de cargar una imagen de sistema operativo en la memoria de la computadora y comenzar la ejecución del sistema operativo.

Se trata de un tema raramente discutido en libros de sistemas operativos.

El proceso de booteo es altamente dependiente de la máquina. Aquí se considera solamente el proceso de booteo de las PCs basadas en Intel x86.

Booteo de las PCs basadas en Intel x86

Toda PC tiene un programa BIOS (Basic Input Output System) almacenado en ROM (Read Only Memory).

Cuando energiza la PC, o después de un reset, la CPU de la PC empieza a ejecutar el BIOS. El BIOS primero realiza el POST (Power-on Self Test) para revisar que el hardware del sistema funciona apropiadamente.

Después el BIOS busca un dispositivo desde el cual llevar a cabo el proceso de booteo. Los dispositivos de booteo están almacenados en una memoria CMOS programable.

El orden de booteo usual es disco floppy, CDROM, disco duro.

Si el BIOS encuentra un dispositivo bootable, trata de bootear desde ese dispositivo. En caso contrario, despliega un mensaje: "no bootable device found".

Prezi

Dispositivos de arranque (Bootable Devices)

Un dispositivo de arranque es un dispositivo de almacenamiento soportado por el BIOS para el proceso de booteo.

Actualmente, entre los dispositivos de arranque se tienen:
Disco floppy
Disco duro
Disco CD/DVD
Unidad USB

Un dispositivo bootable contiene un booter y una imagen de sistema bootable. Durante el booteo, el BIOS carga los primeros 512 bytes del booter a la ubicación de memoria (segment,offset)=(0x0000,0x7C00)=0x07C00, y salta ahí para ejecutar el booter.
Después de eso, es completamente responsabilidad del booter hacer el resto.

Prezi

Dispositivos de arranque (Bootable Devices) (Cont.)

El BIOS siempre carga el booter en 0x07C00 por razones históricas: En las primeras PC solo se tenía la garantía de tener 64 KB de memoria RAM. La memoria debajo de 0x07C00 está reservada para vectores de interrupción, BIOS y BASIC, etc.

La primera memoria utilizable para el sistema operativo comienza en 0x08000. Así que el booter es cargado a 0x07C00, lo cual está 1 KB abajo de 0x08000.

Cuando la ejecución comienza, las acciones del booter típicamente son:
Cargar el resto del booter a la memoria y ejecutar el booter completo.

Encontrar y cargar la imagen del sistema operativo a la memoria.
Hacer que la CPU ejecute el código de inicialización del kernel del sistema operativo, el cual inicia el sistema operativo.

Prezi

Floppy Disk Booting

Como dispositivo de arranque, el disco floppy (FD) está casi obsoleto. A pesar de eso, vale la pena discutir FD booting por varias razones:

Es mucho más seguro escribir al sector 0 de un FD que al sector 0 (MBR) de un disco duro.

FD proporciona una forma simple y segura para aprender el proceso de booteo y probar nuevos programas booter.

Casi todos los emuladores de PC (QEMU, VMware, Virtualbox) aun soportan unidades floppy.

Estos emuladores de PC proporcionan un ambiente de máquina virtual para desarrollar y probar software de sistema.

Prezi

Casos de arranque desde disco floppy

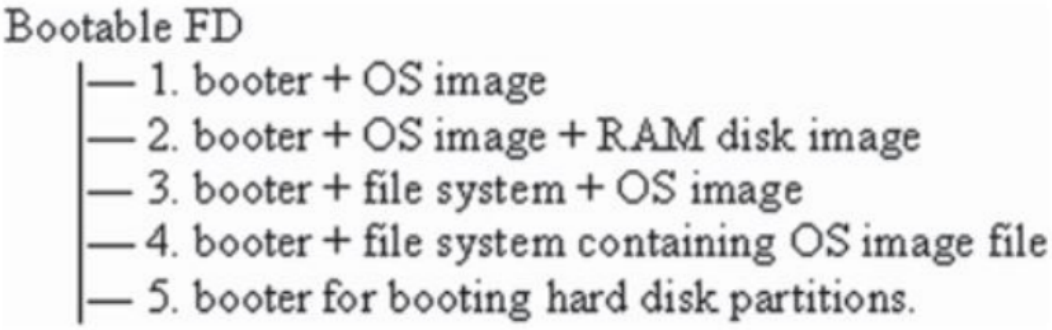


Fig. 3.1 Bootable FDs by Contents

Prezi

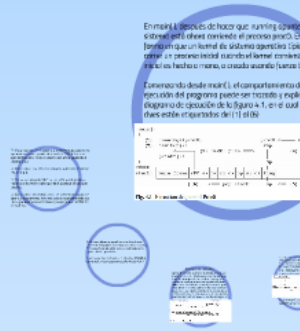
Desarrollo del kernel MTX

Desarrollo de un sistema multitarea simple

A través de este ejercicio práctico se mostrarán dos puntos importantes:

- Cómo comienzan los procesos en un sistema operativo
- Cómo un sistema operativo corre diferentes procesos

Referencia: [1] D&I of the MTX OS, Sec. 4.3, página 99 (116/551) .



Archivos para el sistema MTX 4.0

Los archivos necesarios para construir el sistema MTX 4.0 son los siguientes:
Para el cargador de arranque (booter):
MTX4_0/mtxFS/bs.s
MTX4_0/mtxFS/bc.c
MTX4_0/mtxFS/ext2.h
Para el kernel (mtx):
MTX4_0/ts.s
MTX4_0/t.c
Para el booter se usará la biblioteca
/usr/lib/bcc/libc.a
Para el kernel se usará la biblioteca
MTX4_0/mtxlib
La biblioteca /usr/lib/bcc/libc.a se instala con el paquete elks-libc. Para nuestros propósitos, es suficiente con instalar el paquete debian bcc. También debemos instalar el paquete debian genext2fs. El archivo de biblioteca mtxlib lo puede descargar del directorio MTX4_0 del repositorio
https://github.com/sqrtpeacher/dirty-work/

Construcción del cargador de arranque

El cargador de arranque es un programa que se encarga de colocar el kernel en la memoria y tranferir el control de la ejecución al mismo.

Construcción del cargador de arranque (booter)
El cargador de arranque se construye a partir de los archivos
mtxFS/bs.s y mtxFS/bc.c
os86 -o bs.o mtxFS/bs.s
bcc -o bc.o -c -ansi mtxFS/bc.c
ld86 -d -o booter bs.o bc.o /usr/lib/bcc/libc.a

Ejecución del kernel MTX 4.0

Para ejecutar el kernel MTX 4.0 creamos una imagen de disco floppy con sistema de archivos ext2, colocando el kernel en el directorio boot/ . Para correr el sistema utilizamos el programa qemu-system-i386 con las opciones indicadas.

Ejecución del kernel MTX 4.0
cp -v mtx /mount_point/boot/
genext2fs -v -b 1440 -d ./mount_point/ mFDimage
dd if=booter of=mFDimage bs=1024 count=1 conv=notrunc
qemu-system-i386 -fda mFDimage -no-fd-bootchk
(si está usando la app Debian sobre Windows, no olvide que debe estar ejecutando el servidor Xming, y que previamente se debe exportar la variable DISPLAY:
export DISPLAY=:0
)

Prezi

Archivos para el sistema MTX 4.0

Los archivos necesarios para construir el sistema MTX 4.0 son los siguientes:
Para el cargador de arranque (booter):
MTX4_0/mtxFS/bs.s
MTX4_0/mtxFS/bc.c
MTX4_0/mtxFS/ext2.h
Para el kernel (mtx):
MTX4_0/ts.s
MTX4_0/t.c
Para el booter se usará la bilioteca
/usr/lib/bcc/libc.a
Para el kernel se usará la biblioteca
MTX4_0/mtxlib
La biblioteca /usr/lib/bcc/libc.a se instala con el paquete elks-libc. Para nuestros propósitos, es suficiente con instalar el paquete debian bcc. También debemos instalar el paquete debian genext2fs. El archivo de biblioteca mtxlib lo puede descargar del directorio MTX4_0 del repositorio https://github.com/sotrteacher/dirtywork/

Prezi

Construcción del cargador de arranque

El cargador de arranque es un programa que se encarga de colocar el kernel en la memoria y tranferir el control de la ejecución al mismo.

Construcción del cargador de arranque (booter)
El cargador de arranque se construye a partir de los archivos mtxFS/bs.s y mtxFS/bc.c

```
as86  -o bs.o  mtxFS/bs.s
bcc  -o bc.o  -c -ansi  mtxFS/bc.c
ld86  -d  -o booter  bs.o  bc.o  /usr/lib/bcc/libc.a
```

Prezi

Construcción del kernel MTX 4.0

El kernel MTX 4.0 es el primero de los kernel del sistema operativo MTX presentados en el capítulo 4 del libro [1] Design and implementation of the MTX Operating System.

Construcción del kernel MTX 4.0
El programa MTX 4.0 es similar a un booter. Lo compilaremos y enlazaremos usando BCC para generar un binario ejecutable

```
as86 -o ts.o ts.s          # ensamblar ts.s
bcc -c -ansi t.c           # compilar t.c
ld86 -d -o mtx  ts.o  t.o  mtxlib  /usr/lib/bcc/libc.a    # enlazar
```

Prezi

Ejecución del kernel MTX 4.0

Para ejecutar el kernel MTX 4.0 creamos una imagen de disco floppy con sistema de archivos ext2, colocando el kernel en el directorio boot/ . Para correr el sistema utilizamos el programa qemu-system-i386 con las opciones indicadas.

Ejecución del kernel MTX 4.0

```
cp -v mtx  ./mount_point/boot/
genext2fs -v -b 1440  -d  ./mount_point/   mFDimage
dd  if=booter  of=mFDimage  bs=1024 count=1  conv=notrunc
qemu-system-i386  -fda  mFDimage  -no-fd-bootchk
```

(si está usando la app Debian sobre Windows, no olvide que debe estar ejecutando el servidor Xming, y que previamente se debe exportar la variable DISPLAY:

```
export DISPLAY=:0
```

)

Prezi

Booteo de MTX 4.0

Durante el arranque, el booter MTX carga el kernel mtx0 al segmento 0x1000 y salta ahí para ejecutar el código de mtx0. Cuando la ejecución comienza en ts.s, este establece todos los registros de segmento a 0x1000 para que estos correspondan al modelo de memoria de un segmento de programa. Entonces establece el apuntador de pila al extremo alto de proc0, así que proc0.kstack es el área de pila inicial.

Hasta este punto, el sistema no tiene noción de algún proceso porque aun no existe proceso alguno. Entonces el código ensamblador llama a main( ) en C. Cuando el control entra a main( ), tenemos una imagen en ejecución. Por la definición de proceso, la cual es la ejecución de una imagen, tenemos un proceso en ejecución, aunque el sistema aun no sabe cuál proceso está ejecutando.

Prezi

En main( ), después de hacer que running apunte a proc0, el sistema está ahora corriendo el proceso proc0. Esta es la forma en que un kernel de sistema operativo típico empieza a correr un proceso inicial cuando el kernel comienza. El proceso inicial es hecho a mano, o creado usando fuerza bruta.

Comenzando desde main( ), el comportamiento de tiempo de ejecución del programa puede ser trazado y explicado por el diagrama de ejecución de la figura 4.1, en el cual los pasos clves están etiquetados del (1) al (6)

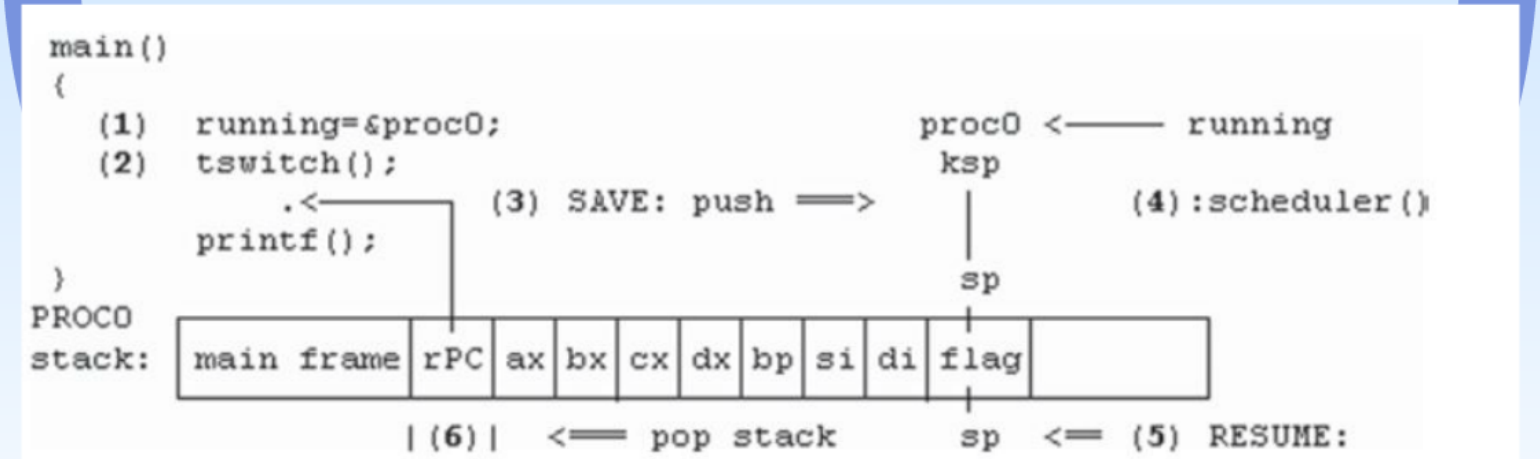


Fig. 4.1 Execution diagram of Proc0

En (1), se hace que running apunte a proc0. Dado que suponemos que running siempre apunta a la estructura PROC del proceso ejecutándose actualmente, el sistema está ahora ejecutando el proceso proc0.

En (2), se llama a tswitch( ), el cual guarda la dirección de retorno, rPC, en la pila.

En (3), se ejecuta la parte SAVE de tswitch( ), lo cual guarda los registros de la CPU en la pila y guarda el apuntador de pila sp en pro0.ksp.

En (4), se llama a scheduller( ), el cual establece running para que apunte a proc0 otra vez. Por ahora, esto es redundante dado que running ya apunta a proc0. Entonces, se ejcuta la parte RESUME de tswitch( ).

Prezi

En (5), se establece sp a proc0.ksp, lo cual es otra vez redundante dado que ambos ya contienen el mismo valor. Entonces se hace pop sobre la pila, lo cual restaura los registros de CPU guardados.

En (6), se ejecuta la instrucción ret al final de RESUME, lo cual regresa al lugar desde donde se llamó a tswitch( ).

Prezi

Cambio de contexto

Aparte del hecho de imprimir unos cuantos mensajes, el programa parece inutil dado que no hace practicamente nada. Sin embargo, es la base de todos los programas multitarea. Supongamos que tenemos otra estructura PROC, proc1, la cual llamó a tswitch( ) y ejecutó la parte SAVE de tswitch( ) antes. Entonces el ksp de proc1 debe apuntar a su área de pila, la cual contiene los registros de la CPU guardados y una dirección de retorno desde donde proc1 llamó a tswitch( ), como se muestra en la figura 4.2.

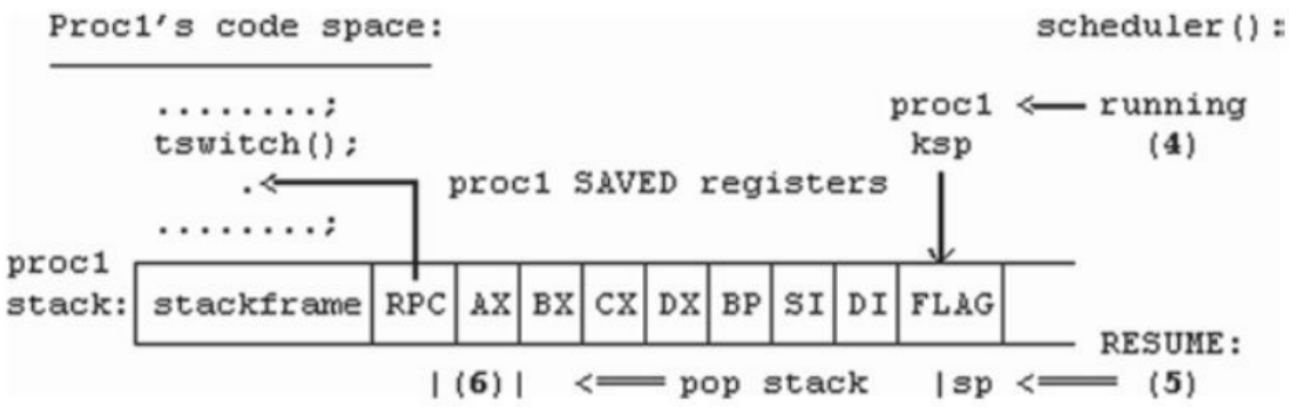


Fig. 4.2 Execution diagram of Proc1

Prezi

Si en scheduler( ) permitimos que running apunte a proc1, como se muestra en el lado derecho de la figura 4.2, la parte RESUME de tswitch( ) cambiaría sp al valor del ksp de proc1. Entonces el código RESUME operaría sobre la pila de proc1. Esto restauraría los registros guardados de proc1, causando que proc1 regrese a ejecución desde donde llamó a tswicth( ) antes. Esto cambia el ambiente de ejecución del de proc0 al de proc1.

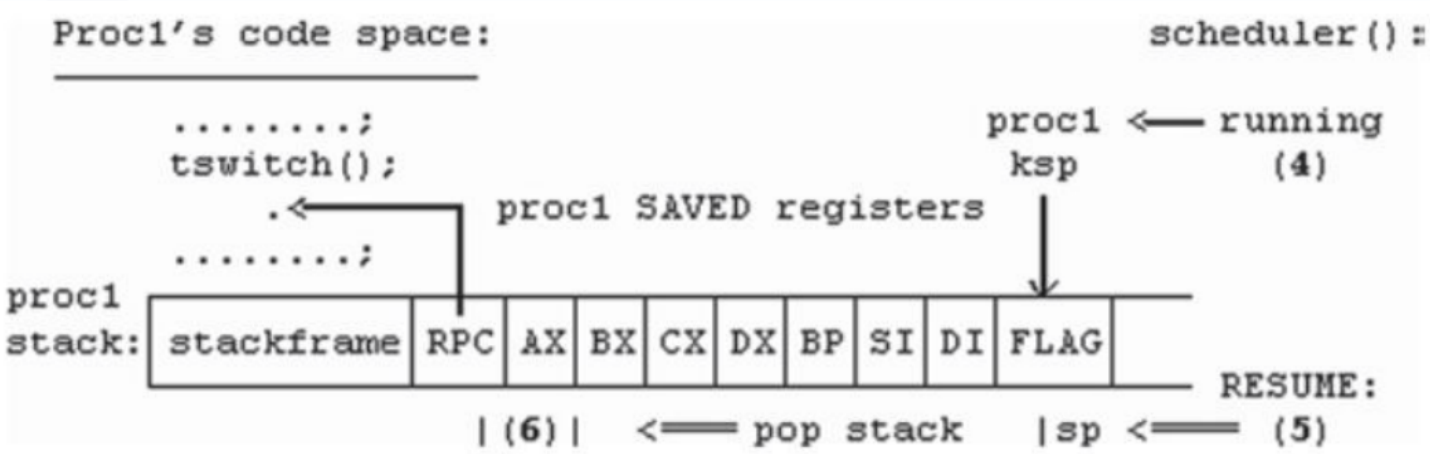


Fig. 4.2 Execution diagram of Proc1

Cambio de contexto El cambio del ambiente de ejecución de un proceso al de otro proceso es llamado cambio de contexto, lo cual es el mecanismo básico de la multitarea.

Prezi

UT I Introducción a los sistemas operativos en tiempo real

Analiza un sistema operativo en tiempo real en base a los sistemas operativos existentes

Prezi