21/12/2018

|  |
| --- |
| Convocatoria de Enero | Luis Bleda Torres, Samuel Sánchez Álvarez |



|  |  |
| --- | --- |
| XV6 | AMPLIACIÓN DE SISTEMAS OPERATIVOS |

Contenido

[Boletín 7: Implementación de llamadas al sistema 3](#_Toc533003002)

[Ejercicio 1 3](#_Toc533003003)

[Ejercicio 2 4](#_Toc533003004)

[Boletín 8: Reserva de páginas bajo demanda 5](#_Toc533003005)

[Ejercicio 1 5](#_Toc533003006)

[Ejercicio 2 7](#_Toc533003007)

[Boletín 9: Ficheros grandes en XV6 8](#_Toc533003008)

[Ejercicio 1 8](#_Toc533003009)

[Ejercicio 2 9](#_Toc533003010)

# Boletín 7: Implementación de llamadas al sistema

## Ejercicio 1

Añadimos en syscall.h un nuevo número a la llamada al sistema date. En usys.S añadimos la llamada date

SYSCALL(date)

#define SYS\_date 22 //Codigo de la llamada al sistema date…

Seguidamente en syscall.c añadimos la definición de la función sys\_date(),y en el array de funciones (\*syscalls[])(void) añadimos la nueva entrada

Ahora procedemos a la implementación de la llamada al sistema. Para ello, vamos al archivo sysproc.c, donde se sitúan las llamadas al sistema relacionadas con procesos. Como la llamada al sistema no es referente sobre el sistema de ficheros la añadimos aquí.

extern int sys\_date(void);

static int (\*syscalls[])(void) = {

…

[SYS\_date] sys\_date,

};

//Llamada al sistema para mostrar la fecha

int

sys\_date(void)

{

struct rtcdate \*r;

if (argptr(0,(char\*\*)&r,sizeof(struct rtcdate))<0)

return -1;

//Cogemos de la pila de usuario

cmostime(r); //Mostramos la fecha

return 0;

}

## Ejercicio 2

Procedemos de la misma manera que con la llamada date: Añadimos en syscall.h un nuevo número a la llamada al sistema dup2. En usys.S añadimos la llamada dup2

SYSCALL(dup2)

#define SYS\_dup2 23 //Codigo de la llamada al sistema dup2

Seguidamente en syscall.c añadimos la definición de la función sys\_dup2() y en el array de funciones (\*syscalls[])(void) añadimos la nueva entrada

extern int sys\_dup2(void);

static int (\*syscalls[])(void) = {

…

[SYS\_dup2] sys\_dup2,

};

Añadimos la implementación en el fichero sysfile.c

# Boletín 8: Reserva de páginas bajo demanda

//Llamada al sistema para implementar dup2

int

sys\_dup2(void)

{

struct file \*f1;

struct file \*f2;

int fd,fd2;

struct proc \*proc=myproc();

//Cogemos los descriptores de fichero

if(argfd(0,&fd,&f1)<0)

return -1;

if(argint(1,&fd2)<0)

return -1;

if(fd2<0 || fd2 >= NOFILE)

return -1;

//Comprobamos si son iguales

if(fd==fd2)

return fd2;

if ((f2=proc->ofile[fd2])!=0) //Si el descriptor está abierto, hay que cerrarlo

fileclose(proc->ofile[fd2]);

proc->ofile[fd2]=f1; //Indicamos que el descriptor nuevo apunta al viejo

filedup(f1); //Hacemos la llamada dup

return fd2;

}

## Ejercicio 1

Para implementar la reserva de páginas bajo demandas debemos modificar la llamada al sistema sys\_sbrk() para que sólo aumente el tamaño del proceso, y en caso de necesitarlo, el sistema operativo solicite otra página.

int

sys\_sbrk(void)

{

int addr;

int n;

if(argint(0, &n) < 0) //Comprobamos si se nos pasan un argumento de tipo entero

return -1;

addr=myproc()->sz;

if(n<0){ //Si es negativo llamamos a growproc

if(growproc(n) < 0)

return -1;

return addr;

}

if(addr+n >= KERNBASE) //Si nos pasamos y ocupamos el mapa del kernel es un error

return -1;

myproc()->sz+=n; //Aumentamos el tamaño del proceso

return addr;

}

Además de esto, hay que responder al fallo de página mapeando una nueva página física y volver a modo usuario para que el proceso continúe. Modificamos la función trap() del fichero trap.c para que, en caso de fallo de página, le dé una nueva página.

// In user space, actualizamos tabla de paginas

mem = kalloc();

if(mem == 0){

cprintf("out of memory inside page fault\n");

myproc()->killed=1;

break;

}

memset(mem, 0, PGSIZE);

if(mappages(myproc()->pgdir, (char\*) PGROUNDDOWN(rcr2()), PGSIZE, V2P(mem), PTE\_W|PTE\_U) < 0){

cprintf("mappages error inside page fault\n");

kfree(mem);

myproc()->killed=1;

}

## Ejercicio 2

* El caso de un argumento negativo al llamarse a sbrk()

Cuando el argumento es negativo, llamamos a growproc que se encarga de gestionar esa situación.

if(n<0){ //Si es negativo llamamos a growproc

if(growproc(n) < 0)

return -1;

return addr;

}

* Manejar el caso de fallos en la página inválida debajo de la pila

En la función mappages()  del fichero vm.c modificamos ese caso

* Verificar que fork() y exit()/wait()funciona en el caso de que haya direcciones virtuales sin memoria reservada para ellas

Modificamos la función copyuvm() de forma similar a como lo hicimos con mappages()

if(\*pte & PTE\_P) //Si hay fork o exit/wait y se sale fuera del proceso lo consideramos como una situación normal

return 0;

Ilustración: Modificación tanto a mappages como a copyuvm

* Asegurarse de que funciona el uso por parte del kernel de páginas de usuario que todavía no han sido reservadas (p.e., si un programa pasa una dirección de la zona de usuario todavía no reservada a read())

En trap() cambiamos la condición para que cuando el kernel solicite páginas de usuario no sea considerado un error.



# Boletín 9: Ficheros grandes en XV6

Ejercicio 1

Modificamos los valores de NDIRECT en el fichero fs.h y MAXFILE

#define NDIRECT 11

#define MAXFILE (NDIRECT + NINDIRECT+NINDIRECT\*NINDIRECT)

En dinode (el nodo-i en disco, implementado en fs.h) y en inode (el nodo-i cargado en memoria, implementado en file.h) modificamos el array addrs que representan los bloques del nodo-i para indicar que ahora hay 11 bloques directos, 1 BSI y 1 BDI

// in-memory copy of an inode

struct inode {

…

uint addrs[NDIRECT+1+1];

};

// On-disk inode structure

struct dinode {

…

uint addrs[NDIRECT+1+1]; // Data block addresses

};

Modificamos la función bmap() en el fichero fs.c para que ahora, además de un BSI, añada un BDI.

// Return the disk block address of the nth block in inode ip.

// If there is no such block, bmap allocates one.

static uint

bmap(struct inode \*ip, uint bn)

{

//Está en el BDI, asi que lo añadimos

bn-=NINDIRECT;

if(bn < NINDIRECT\*NINDIRECT){

if((addr = ip->addrs[NDIRECT+1]) == 0)

ip->addrs[NDIRECT+1] = addr = balloc(ip->dev);

uint direct\_block=bn%NINDIRECT; //Posicion del bloque directo del BSI del BDI

bn/=NINDIRECT; //Posicion del bloque BSI del BDI

bp=bread(ip->dev,addr); //Leemos el BDI

a=(uint\*)bp->data;

if((addr=a[bn])==0){

a[bn]=addr=balloc(ip->dev);

log\_write(bp);

}

brelse(bp);

bp=bread(ip->dev,addr); //Leemos el BSI

a=(uint\*)bp->data;

if((addr=a[direct\_block])==0){ //Leemos el bloque directo del BDI

a[direct\_block]=addr=balloc(ip->dev);

log\_write(bp);

}

brelse(bp);

return addr;

}

## Ejercicio 2

Modificamos la función itrunc() en el fichero fs.c

// Truncate inode (discard contents).

// Only called when the inode has no links

// to it (no directory entries referring to it)

// and has no in-memory reference to it (is

// not an open file or current directory).

static void

itrunc(struct inode \*ip)

{

int i, j,k,l;

struct buf \*bp;

uint \*a;

…

if(ip->addrs[NDIRECT+1]){

bp=bread(ip->dev,ip->addrs[NDIRECT+1]); //Leemos el BDI

a=(uint\*) bp->data;

for(k=0; k < NINDIRECT;k++){ //Recorremos el BDI

if(a[k]){ //Si la posicion k no está vacía (hay bloque) recorremos el BSI de esa posicion

struct buf \*auxiliar=bread(ip->dev,a[k]); //Leemos el BSI

uint\* b=(uint\*)auxiliar->data;

for(l=0;l<NINDIRECT;l++){ //Recorremos el BSI

if(b[l]) //Si esa posicion no esta vacia, liberamos directamente

bfree(ip->dev,b[l]);

}

brelse(auxiliar); //Liberamos el bloque

bfree(ip->dev,a[k]); //Liberamos el BSI

}

}

//Liberamos el BDI y terminamos

brelse(bp);

bfree(ip->dev, ip->addrs[NDIRECT+1]);

ip->addrs[NDIRECT+1] = 0;

}

ip->size = 0;

iupdate(ip);

}