TP1

lunes, 4 de abril de 2022 17:55

Objetivos:

- Modificar, compilar y testear el sistema operativo Xinu.
- Crear procesos, finalizar procesos.
- Conocer los componentes de la tabla de procesos en Xinu.

La versión de Xinu que utilizaremos es para arquitectura PC (x86). Utilizaremos el sistema operativo Xinu utilizando una máquina virtual llamada QEMU, que emula una PC básica.
El trabajo puede realizarse sobre las máquinas de los laboratorios (RECOMENDADO).
Quienes tengan Linux en sus casas, podrían intentar instalar todo lo necesario y llevarlo a cabo ahí también.

Ejercicio01

lunes, 4 de abril de 2022 17:56

Descargue el código fuente de Xinu para PC, desde la web de la materia. El sistema operativo ya contiene, también, el código fuente del shell de Xinu.

Compilar y verificar el funcionamiento de Xinu utilizando las instrucciones en la web de la materia.

- ¿Qué componentes trae esta versión de Xinu?
- ¿Qué periféricos de PC son accesibles desde QEMU?
- ¿Cómo se accede al puerto serie de la PC en QEMU, el cual permite utilizar el shell?
- ¿Cuántos procesos existen en ejecución? ¿Cómo lo obtuvo?

Los componentes que trae esta versión de Xinu son

- Administrador de memoria
- Administrador de procesos
- Sincronización y coordinación de procesos
- Comunicación interprocesos
- Administrador de reloj en tiempo real
- Dispositivo de entrada y salida independiente
- Dispositivo de controladores
- Protocolos de red
- Sistema de archivos

Los periféricos que son accesibles desde QEMU son el puerto VGA, PS2 y SERIE.

Para poder acceder al puerto serie de la PC en QEMU hay usar el atajo de teclado Ctrl + Alt + 3.

Para saber cuántos procesos hay en ejecución hay que ejecutar el comando ps.

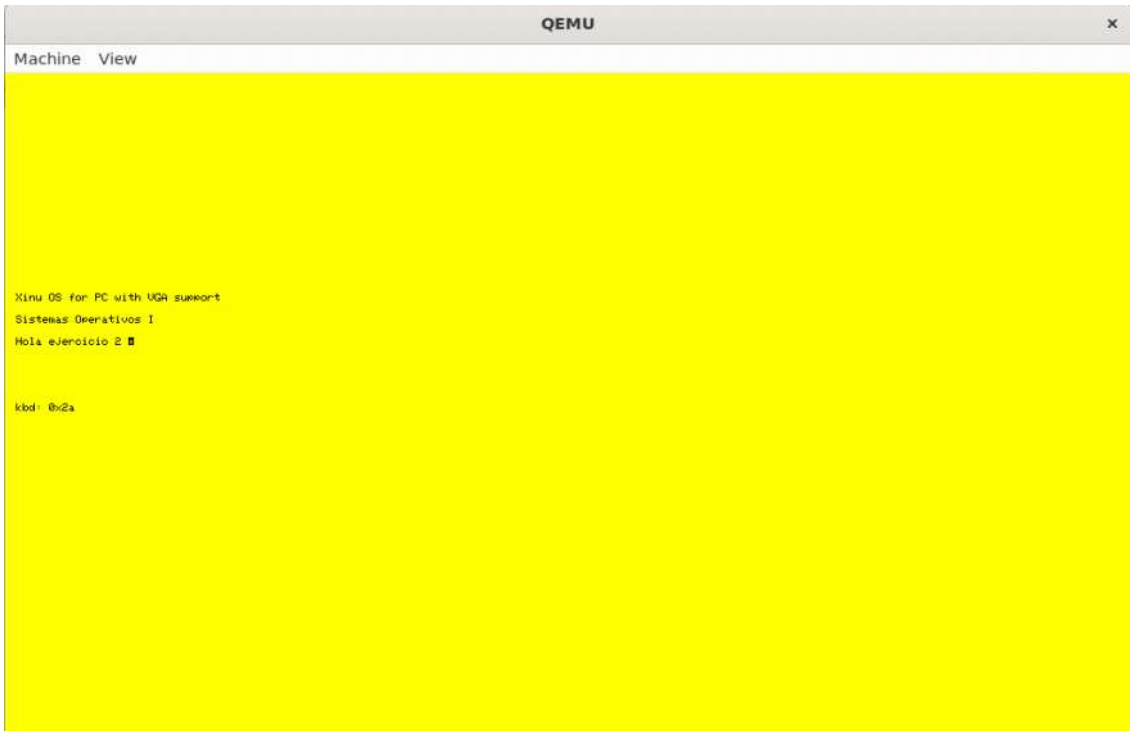
Ejercicio02

lunes, 4 de abril de 2022 18:11

Modifique Xinu. Editar el archivo main.c y emita un mensaje a la pantalla gráfica a coleres VGA de PC. Compilar y verificar.

Archivo system/main.c

```
/*  main.c  - main */
#include <xinu.h>

extern paint_screen();
extern print_text_on_vga(unsigned int x, unsigned int y, const char * t);
process main(void)
{

    paint_screen();
    print_text_on_vga(10, 200, "Xinu OS for PC with VGA support");
    print_text_on_vga(10, 220, "Sistemas Operativos I");

    // linea agregada para el ejercicio 2
    print_text_on_vga(10, 240, "Hola ejercicio 2 \n");

    recvclr();
    resume(create(shell, 8192, 50, "shell", 1, CONSOLE));
    /* Wait for shell to exit and recreate it */
    while (TRUE) {
        receive();
        sleepms(200);
        kprintf("\n\nMain process recreating shell\n\n");
        resume(create(shell, 4096, 20, "shell", 1, CONSOLE));
    }
    return OK;

}
```



Ejercicio03 - Incorporación de un programa al shell de Xinu

lunes, 4 de abril de 2022 18:19

Ejercicio 3. *Incorporación de un programa al shell de Xinu.*

Agregue un nuevo programa a Xinu. Deberá incorporarlo al shell de Xinu de la siguiente manera:

1. Escriba un programa hello world en un archivo .c debajo del directorio **shell/**. Atención, en Xinu la función principal de un programa no debe llamarse main(). Ejemplo:

```
#include <xinu.h>

mi_programa() {
        printf("Hola mundo! \n");
}
```

2. Incorpore su programa a la estructura que define los programas del shell, dentro del archivo **shell/cmdtab.c** y en **include/shprototypes.h**
3. Compilar y verificar que su programa se encuentra incorporado al sistema y puede ejecutarlo.
4. ¿Por qué en Xinu no se llama main() la función principal de cada nuevo programa?

Nombre del archivo: mi_programa.c

shell/cmdtab.c

```
 9  const   struct  cmdent  cmdtab[] = {
10      {"argecho", TRUE,   xsh_argecho},
11      {"arp",     FALSE,  xsh_arp},
12      {"cat",     FALSE,  xsh_cat},
13      {"clear",   TRUE,   xsh_clear},
14      {"date",    FALSE,  xsh_date},
15      {"devdump", FALSE,  xsh_devdump},
16      {"echo",    FALSE,  xsh_echo},
17      {"exit",    TRUE,   xsh_exit},
18      {"help",    FALSE,  xsh_help},
19      {"kill",    TRUE,   xsh_kill},
20      {"memdump", FALSE,  xsh_memdump},
21      {"memstat", FALSE,  xsh_memstat},
22      {"netinfo", FALSE,  xsh_netinfo},
23      {"ping",    FALSE,  xsh_ping},
24      {"ps",      FALSE,  xsh_ps},
25      {"sleep",   FALSE,  xsh_sleep},
26      {"udp",     FALSE,  xsh_udpdump},
27      {"udpecho", FALSE,  xsh_udpecho},
28      {"udpeserver",  FALSE,  xsh_udpeserver},
29      {"uptime",  FALSE,  xsh_uptime},
30      {"?",       FALSE,  xsh_help},
31      {"programa",    FALSE,  mi_programa}
32  };
33
34  uint32  ncmd = sizeof(cmdtab) / sizeof(struct cmdent);
```

```
32    };
33
34    uint32  ncmd = sizeof(cmdtab) / sizeof(struct cmdent);
```

Archivo includo/shprototypes.c

```
79    /* in file mi_programa.c */
80    extern shellcmd mi_programa      (void);
81
```

Salida por pantalla:

```
xsh $ programa
Hola mundo!
xsh $ help

shell commands are:

argecho      date         help         netinfo      udp          ?
arp          devdump      kill         ping         udpecho      programa
cat          echo         memdump      ps           udpeserver
clear        exit         memstat      sleep        uptime
xsh $
```

4. No se llama main la función principal del programa porque el nombre del archivo debe coincidir con la función principal, pero como el archivo system/main.c ya existe no es posible usar ese nombre.

Ejercicio04 - Creación de procesos en Xinu

lunes, 4 de abril de 2022 18:26

Incorpore a su programa anterior código para crear los siguientes dos procesos que se observan en el ejemplo:
https://github.com/zrafa/xinu/blob/main/xinu-pc/misc/ej1.c

- ¿Qué sucede al ejecutar el programa?
- ¿Cómo se podría lograr que el sistema operativo conmute la CPU entre los dos procesos de manera más seguida?. **Investigar qué es el QUANTUM, de manera general, para los sistemas operativos, y para qué lo utilizan los sistemas operativos. Averiguar cómo es posible modificarlo en Xinu.**

Al ejecutar el programa se crean dos procesos en el que uno imprime Aes y el otro imprime Bes infinitamente, en la consola se muestran infinitas As y Bs de manera muy rápida.

El quantum es el tiempo máximo que un proceso puede hacer uso del procesador con el algoritmo de planificación de procesos Round-Robin. Puede ser fijo o variable y puede tener el mismo valor para todos los procesos o distinto.
El quantum de un proceso equivalente a un número fijo de pulsos o ciclos de reloj. Al ocurrir una interrupción de reloj que coincide con la agitación del quantum se llama al despachador, el cual le cede el control de la CPU al procesos seleccionado por el planificador.

Para modificar el quantum en xinu hay que buscar la variable global llamada QUANTUM en el archivo include/kernel.c y cambiarle el valor que está en milisegundos.

Ejercicio05 - finalización de procesos en Xinu

lunes, 4 de abril de 2022 19:09

Incorpore a main, luego de creado los procesos del Ejercicio 4, una espera de 10 segundos. Y que luego de la espera finalice los dos procesos iniciados anteriormente. (Ayuda: almacenar los PIDs de los procesos y averiguar cómo se llama la system call para finalizar procesos).

```
/* ex2.c - main, sndA, sndB */
#include <xinu.h>
void    sndA(void), sndB(void);
/*------------------------------------------------------------------------
 * main  --  example of creating processes in Xinu
 *------------------------------------------------------------------------
 */
void    mi_programa(void)
{
    int idp1 = create(sndA, 1024, 20, "process 1", 0);
    int idp2 = create(sndB, 1024, 20, "process 2", 0);
    resume( idp1 );
    resume( idp2 );
    sleep(10);
    int pid = getpid();
    kill(idp1);
    kill(idp2);
    printf("pid %i \n", pid);
}
/*------------------------------------------------------------------------
 * sndA  --  repeatedly emit 'A' on the console without terminating
 *------------------------------------------------------------------------
 */
void    sndA(void)
{
    while( 1 ) {
        //sleep(1);
        putc(CONSOLE, 'A');
    }
}
/*------------------------------------------------------------------------
 * sndB  --  repeatedly emit 'B' on the console without terminating
 *------------------------------------------------------------------------
 */
void    sndB(void)
{
    while( 1 ) {
        //sleep(1);
        putc(CONSOLE, 'B');
    }
}
```

Ejercicio06 - Varios procesos reutilizando código ejecutable

lunes, 4 de abril de 2022 19:14

Incorpore a su programa código para crear los siguientes dos procesos que se observan en el siguiente ejemplo: https://github.com/zrafa/xinu/blob/main/xinu-pc/misc/ej2.c

En este caso es igual al programa anterior solo que se le envía los caracteres que tiene que imprimir como argumentos de la función que tiene que ejecutar cada proceso.

```c
/* ex3.c - main, sndch */

#include <xinu.h>

void    sndch(char);

/*------------------------------------------------------------------------
 * main  --  example of 2 processes executing the same code concurrently
 *------------------------------------------------------------------------
 */
void    main(void)
{
        resume( create(sndch, 1024, 20, "send A", 1, 'A') );
        resume( create(sndch, 1024, 20, "send B", 1, 'B') );
}

/*------------------------------------------------------------------------
 * sndch  --  output a character on a serial device indefinitely
 *------------------------------------------------------------------------
 */
void    sndch(char ch)

{

        while ( 1 )
                putc(CONSOLE, ch);
}
```

Ejercicio07

lunes, 4 de abril de 2022 19:17

Ejercicio 7

Utilice el ejemplo de la clase para crear un programa en Linux, que le pida al sistema operativo crear un nuevo proceso hijo.

```c
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>
void main()
{
    /* fork a child process */
    int pid = fork();

    if (pid < 0)
    { /* error occurred */
        printf(stderr, "Fork Field\n");
        return 1;
    }
    else if (pid == 0)
    { /* child process */
        printf("Proceso hijo %d\n", pid);
    }
    else
    { /* parent process */
        printf("Proceso padre. Mi hijo es el pid = %d\n", pid);
        wait(NULL);
        printf("Child Complete\n");
    }

    return 0;
}
```

```
braian@Desktop-Braian:/mnt/c/Users/braia/OneDrive/Estudios/Universidad/LCC/Sistemas Operativos 1/TPs/TP01$ ./ejercicio07
Proceso padre. Mi hijo es el pid = 38
Proceso hijo 0
Child Complete
```

Este programa permite conocer el pid del proceso hijo creado.

Ejercicio08

lunes, 4 de abril de 2022 19:39

Douglas Comer en el libro que documenta la implementación de Xinu dice:
The name stands for Xinu Is Not Unix. As we Will see, the interna/ structure of Xinu differs completely from the internal structure of IJnix (or Linux). Xinu is smaller, more elegant, and easier to understand.

Ahora compare las funciones que se utilizan en Linux y en Xinu para pedirle luego al sistema crear procesos. ¿Cuál es más fácil en su opinión?, ¿las funciones para realizar system calls en Linux o en Xinu? Si su respuesta fue Linux, entonces ¿por qué piensa que Douglas Comer diría que Xinu es más elegante y más fácil de entender?

RTA: en mi opinión el sistema que es más fácil para crear procesos es linux porque no es necesario saber cuánta memoria va a utilizar y los otros parámetros que pide la función create() en Xinu. Podría sé que Duglas piensa que es más elegante porque es más personalizada la creación de procesos, no hay que manejar el pid del proceso para saber cuál es el que se está ejecutando y decirle que código ejecutar; sino que se le manda la función directamente.

Ejercicio09

lunes, 4 de abril de 2022 19:42

Observar el archivo ***include/process.h*** . Que campos contiene la tabla de procesos en Xinu.
¿Piensa que falta algo más que deba contemplar el sistema operativo para gestionar un proceso?

Para la tabla de procesos Xinu crea la siguiente esstructura:

```
struct procent { /* Entry in the process table */
	uint16 prstate; /* Process state: PR_CURR, etc. */
	pri16 prprio; /* Process priority */
	char *prstkptr; /* Saved stack pointer */
	char *prstkbase; /* Base of run time stack */
	uint32 prstklen; /* Stack length in bytes */
	char prname[PNMLEN]; /* Process name */
	sid32 prsem; /* Semaphore on which process waits */
	pid32 prparent; /* ID of the creating process */
	umsg32 prmsg; /* Message sent to this process */
	bool8 prhasmsg; /* Nonzero iff msg is valid */
	int16 prdesc[NDESC]; /* Device descriptors for process */
};
```

A Xinu le falta almacenar los registros del proceso en el PCB, pero indagando un poco este los guarda en la pila del mismo.

Ejercicio10 - Portar una aplicación entre sistemas operativos

lunes, 4 de abril de 2022 19:44

Portar el juego del ahorcado al sistema operativo Xinu. Agregue un acceso al shell para poder ejecutarlo.
¿Qué modificaciones fue necesario realizar para portar el juego?

AYUDA O: system() es una función de la biblioteca en Linux, no se encuentra en Xinu. Averiguar sobre una equivalente.
AYUDA 1: si utilizó alguna otra función de biblioteca (permitida) entonces averiguar su equivalente en Xinu (si existe).

La modificación que tuve que hacer para que logre compilar el juego fue cambiar la manera en que el programa entra en modo RAW con *contol(0, TC_MODER, 0,0)* en la línea 11. y quitar las que tenía para Linux. Además de modificar los *include* de las bibliotecas por las de Xinu.

```
#include <xinu.h>
const int MAX_LENGHT = 80;
char palabra[80];
int tamanioPalabra = 0;
char palabraAhorcado[80];
int cantFallos = 0;
int MAX_FALLOS = 6;
int ahorcado()
{
    //  modo input en raw
    control(0, TC_MODER,0 ,0);
    solicitarPalabra();
    jugar();
}
void solicitarPalabra()
{
    char c;
    int i = 0;
    while (i < MAX_LENGHT - 1)
    {
        printf("\r                                                                                ");
        printf("\rIngrese la palabra del ahorcado: %s", palabra);
        c = getchar();
        if ((c >= 65 && c <= 90) || (c >= 97 && c <= 122))
        { // si la letra es valida
            palabra[i] = c;
            i++;
        }
        else if (c == 127)
        { // si se borro
            i--;
            if (i < 0)
                i = 0;
            palabra[i] = 0;
        }
        else if (c == 13)
        { // si se ingresa ENTER termina
            tamanioPalabra = i;
            palabra[i] = 0;
            break;
        }
    }

    // copia la palabra en el "pizarron"
    for (int i = 0; i < tamanioPalabra; i++)
    {
        if (i == tamanioPalabra)
            palabraAhorcado[i] = 0;
        else
            palabraAhorcado[i] = '_';
    }
}
/**
 * Verifica si la letra está en la palabra
 * @return int 1 ó 0
 */
int letraAcertada(char c)
{
    int r = 0;
    int i = 0;
    while (r == 0 && i < tamanioPalabra)
    {
        if (palabra[i] == c)
```

```
        {
            r = 1;
        }
        else
        {
            i++;
        }
    }
    return r;
}
void insertarLetra(char c)
{
    for (int i = 0; i < tamanioPalabra; i++)
    {
        if (palabra[i] == c)
        {
            palabraAhorcado[i] = c;
        }
    }
}
int verificarPalabra()
{
    int finalizo = 1;
    int i = 0;
    do
    {
        if (palabra[i] != palabraAhorcado[i])
        {
            finalizo = 0;
            break;
        }
        i++;
    } while (i < tamanioPalabra);
    return finalizo;
}
void jugar()
{
    int c;
    do
    {
        printf("\rPalabra: %s fallos: %i/%i  ingrese una letra (0 para salir): ", palabraAhorcado, cantFallos, MAX_FALLOS);
        c = getchar();
        if ((c >= 65 && c <= 90) || (c >= 97 && c <= 122))
        {
            if (letraAcertada(c))
                insertarLetra(c);
            else
                cantFallos++;
        }
        if (c == '0')
            cantFallos = MAX_FALLOS;
    } while (verificarPalabra() == 0 && cantFallos < MAX_FALLOS);
    printf("\rPalabra: %s fallos: %i/%i  ingrese una letra (0 para salir): ", palabraAhorcado, cantFallos, MAX_FALLOS);
    system("/bin/stty sane");
    printf("\n");
}
```