Sistemas Operativos I
Trabajo Práctico Obligatorio 1

Objetivos:

- Modificar, compilar y testear el sistema operativo Xinu.
- Crear procesos, finalizar procesos (en Xinu y Linux).
- 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.

Ejercicio 1

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.

Mini tutorial de cómo Descargar, compilar y probar Xinu

```
---------------------------------------------------------
# descargar el código fuente
wget https://se.fi.uncoma.edu.ar/so/misc/xinu-pc.tar.gz
# desempaquetar
tar xvf xinu-pc.tar.gz
# compilar
cd xinu-pc/compile/
make clean
make
# ejecutar en una PC (qemu)
make run-qemu
# para acceder a la CONSOLA (shell de XINU)
CTRL+ALT+3
```

- ¿Qué componentes trae esta versión de Xinu?

Esta versión de XINU trae:
Componentes de un sistema operativo, incluidos: mecanismos de gestión de procesos, memoria y temporizadores, facilidades de comunicación entre procesos, funciones de E/S independientes del dispositivo y software de protocolo de Internet.

- ¿Qué periféricos de PC son accesibles desde QEMU?

teclado, mouse, pantalla. Desde QEMU son accesibles:

periférico ps/2
periférico serial
periférico vga

- ¿Cómo se accede al puerto serie de la PC en QEMU, el cual permite utilizar el shell?

Se accede al puerto desde el periférico serial, usando el comando: CTRL+ALT+1
para usar el shell: CTRL+ALT+3

- ¿Cuántos procesos existen en ejecución? ¿Cómo lo obtuvo?

Contando al proceso para obtener la lista de procesos, hay en ejecución 5 procesos. Se obtuvo mediante el comando “ps”.

```
xsh $ ps
Pid Name             State Prio Ppid Stack Base Stack Ptr  Stack Size
--- ---------------- ----- ---- ---- ---------- ---------- ----------
  0 prnull           ready    0    0 0x005FDFFC 0x00127F48     8192
  1 rdsproc          susp   200    0 0x005FBFFC 0x005FBFC8    16384
  3 Main process     recv    20    2 0x005E7FFC 0x005E7F64    65536
  4 shell            recv    50    3 0x005F7FFC 0x005F7C7C     8192
 12 ps               curr    20    4 0x005F5FFC 0x005F5FC4     8192
```

Ejercicio 2
Modifique Xinu. Editar el archivo main.c y emita un mensaje a la pantalla gráfica a colores VGA de PC. Compilar y verificar.
Agregué una instrucción: print_text_on_vga(horizontalpx, verticalpx, string);

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 de código de programa:

```
#include <xinu.h>
mi_programa() {
        printf("Hola mundo! \n");
}
```

Se guardó programa en C (archivo .c) en directorio shell/

2. Incorpore su programa a la estructura que define los programas del shell, dentro del archivo shell/cmdtab.c y en include/shprototypes.h

helloworld agregado a la estructura en shell/cmdtab.c (nombre de la función, debe coincidir con el nombre de archivo)

```
/************************************************************************/
/* Table of Xinu shell commands and the function associated with each   */
/************************************************************************/
const   struct  cmdent  cmdtab[] = {
        {"argecho",     TRUE,   xsh_argecho},
        {"arp",         FALSE,  xsh_arp},
        {"cat",         FALSE,  xsh_cat},
        {"clear",       TRUE,   xsh_clear},
        {"date",        FALSE,  xsh_date},
        {"devdump",     FALSE,  xsh_devdump},
        {"echo",        FALSE,  xsh_echo},
        {"exit",        TRUE,   xsh_exit},
        {"help",        FALSE,  xsh_help},
        {"kill",        TRUE,   xsh_kill},
        {"memdump",     FALSE,  xsh_memdump},
        {"memstat",     FALSE,  xsh_memstat},
        {"netinfo",     FALSE,  xsh_netinfo},
        {"ping",        FALSE,  xsh_ping},
        {"ps",          FALSE,  xsh_ps},
        {"sleep",       FALSE,  xsh_sleep},
        {"udp",         FALSE,  xsh_udpdump},
        {"udpecho",     FALSE,  xsh_udpecho},
        {"udpeserver",  FALSE,  xsh_udpeserver},
        {"uptime",      FALSE,  xsh_uptime},
        {"helloworld",  FALSE,  helloworld},
        {"?",           FALSE,  xsh_help}

};
```

helloworld agregado en include/shprototypes.h (nombre del .c, tambien debe estar en el comment)

```
/* in file xsh_sleep.c */
extern  shellcmd  xsh_sleep     (int32, char *[]);

/* in file xsh_udpdump.c */
extern  shellcmd  xsh_udpdump   (int32, char *[]);

/* in file xsh_udpecho.c */
extern  shellcmd  xsh_udpecho   (int32, char *[]);

/* in file xsh_udpeserver.c */
extern  shellcmd  xsh_udpeserver (int32, char *[]);

/* in file xsh_uptime.c */
extern  shellcmd  xsh_uptime    (int32, char *[]);

/* in file xsh_help.c */
extern  shellcmd  xsh_help      (int32, char *[]);

/* in file helloworld.c */
extern shellcmd helloworld      (int32, char *[]);
```

3. Compilar y verificar que su programa se encuentra incorporado al sistema, y que puede ejecutarlo. Ejecutar qemu con **make run-qemu** en el directorio **xinu-pc/compile**

4. ¿Por qué en Xinu no se llama main() la función principal de cada nuevo programa?
Esto es así pues el programa encargado de mostrar la primera interacción con el usuario lleva ese nombre (main.c), por lo que al compilar otro programa con función principal de nombre "main", lo intentaría sobreescribir, cosa que el compilador no permite.

Al definir una función con el nombre main()

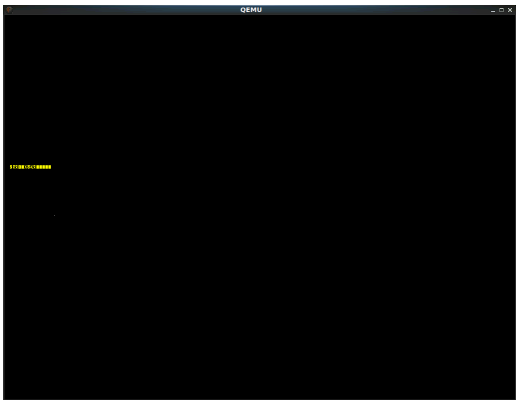
```
Loading object files to produce GRUB bootable xinu
binaries/helloworld.o: En la función `main':
helloworld.c:(.text+0x0): definiciones múltiples de `main'
binaries/main.o:main.c:(.text+0x0): primero se definió aquí
binaries/cmdtab.o:(.data.rel.ro+0xf8): referencia a `helloworld' sin def
Makefile:115: fallo en las instrucciones para el objetivo 'xinu'
make: *** [xinu] Error 1
```

Función main de xinu:

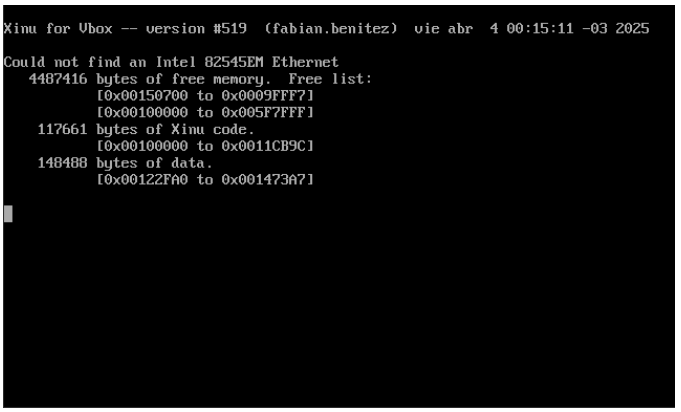
```
/*  main.c  - main */

#include <xinu.h>


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");
        print_text_on_vga(10, 240, "TP01 - Ej02");

        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;

}
```

Resultado de borrar el proceso main() (solo sentencias) de xinu

Al **ejecutar qemu**



Al intentar **mostrar la shell**



Al borrar el archivo main.c

```
Rebuilding the .defs file

Loading object files to produce GRUB bootable xinu
binaries/initialize.o: En la función `startup':
initialize.c:(.text+0x223): referencia a `main' sin definir
Makefile:115: fallo en las instrucciones para el objetivo 'xinu'
make: *** [xinu] Error 1
```

Ejercicio 4. Creación de procesos en Xinu.
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?
Crea dos procesos, uno para imprimir infinitamente la letra ‘A’, y otro igual pero para imprimir la letra ‘B’.

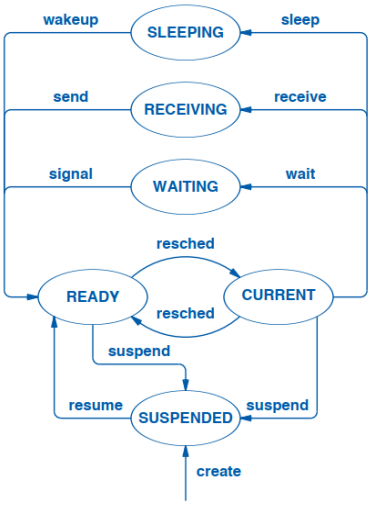
- ¿Cómo se podría lograr que el sistema operativo conmute la CPU entre los dos procesos de manera más seguida?. Investigar qué es el QUANTUM, de manera general, para los sistemas operativos, y para qué lo utilizan los sistemas operativos. Averiguar cómo es posible modificarlo en Xinu.

Esto se podría lograr disminuyendo el quantum del sistema operativo (tiempo que le asigna el CPU a cada proceso).
En XINU se puede cambiar el QUANTUM, modificando el archivo include/kernel.
Párrafo del texto: Operating System Design The Xinu Approach, Douglas Comer
13.6 Implementation Of Preemption
… Defined constant QUANTUM specifies the number of clock ticks in a single time slice. Whenever it switches from one process to another, resched sets global variable preempt to QUANTUM. Each time the clock ticks, the clock interrupt handler decrements preempt. When preempt reaches zero, the clock interrupt handler resets preempt to QUANTUM and calls resched. Following the call to resched, the handler returns from the interrupt.
The call to resched has two possible outcomes. First, if the currently executing process remains the only process at the highest priority, resched will return immediately, the interrupt handler will return, and the current process will return to the point of the interrupt and continue executing for another timeslice. Second, if another ready process has the same priority as the current process, the call to resched will switch to the new process. Eventually, resched will switch back to the interrupted process. The assignment of QUANTUM to preempt handles the case where the current process remains running. The assignment is needed because resched only resets the preemption counter when it switches to a new process.

Ejercicio 5. Finalización de procesos en Xinu.
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). (Douglas Comer)

Capítulo **6.11 Process Termination And Process Exit**, explica el syscall para finalizar procesos.
El syscall es llamado kill(PID)

Capítulo **13.9 Putting A Process To Sleep**, explica el syscall para que un proceso libere el cpu durante un tiempo determinado (ms).
El syscall es llamado sleepms(milisecs)



Ejercicio 6. Varios procesos reutilizando código ejecutable.
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
Modifique el tamaño de la pila al crear cada proceso, dando 8KB de pila a cada uno). Compilar y ejecutar.
Básicamente es modificar el comando create al momento de crear cada proceso, indicando 8KB en lugar de 1024bytes

Ejercicio 7. Creación de procesos en Linux.
Utilice el ejemplo de la clase para crear un programa en Linux, que le pida al sistema operativo crear un nuevo proceso hijo.
El proceso hijo debe obtener los números primos del 1000 al 5000.
El proceso padre (luego de crear el hijo) debe obtener los números primos del 0 al 1000. Cuando finalice debe pedirle al kernel Linux que finalice el proceso hijo. Luego, debe pedirle al sistema operativo finalizar (con exit()). ¿Cuál es el último número primo calculado por el proceso hijo?. ¿Le fue posible calcular todos los que debía calcular?
El último número fue el 1847 (en mi ejecución), no le fue posible calcular todos debido a que el padre (en este caso) terminó antes que el hijo y lo mató antes de que este finalizara, para solucionar esto se podría utilizar un wait luego de que el padre termine su ejecución antes de matar al proceso hijo.

Ejercicio 7-bis
Utilice el ejemplo de la clase para crear un programa en Linux, que le pida al sistema operativo crear un nuevo proceso hijo. El nuevo proceso hijo debe reemplazar su imagen por el del programa /usr/bin/date. El padre debe finalizar realizando una llamada a exit(). ¿Qué funciones de C realizaron llamadas al sistema?

Ejercicio 8
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 internal structure of Xinu differs completely from the internal structure of Unix (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?
Me parece más sencilla la manera en que en Xinu se crean procesos ya que, en mi opinión, es más intuitiva “visualmente” y explícita. En Linux al momento de ejecutar un fork(), se clona implícitamente el main. En cambio, en Xinu se debe explícitamente crear cada proceso y reanudarlo (con resume(pid)).
En cuanto a líneas de código, Linux es más ahorrativo.

Ejercicio 9

Observar el archivo include/process.h. en XINU. ¿Qué campos contiene la tabla de procesos en Xinu?. ¿Piensa que falta algo más que deba contemplar el sistema operativo para gestionar un proceso?
Tabla de procesos Xinu: No se me ocurre que pueda faltar que no se encuentre en la tabla. Tal vez, ¿un contador de procesos hijos?

```
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       */
};
```

Ejercicio 10. Portar una aplicación entre sistemas operativos.
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 0: 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).

Port en base a **mi resolución**: En git y Lab
Port en base a la solución del **profe**:

```
#include <xinu.h>
int ahorcado() {
    int i = 0;
    int c = 0;
    int n = 0;    /* cant de letras */
    char secreta[80];
    char adivina[80];
    int aciertos = 0;
    int vidas = 5;
    int ronda_ok = 0;

    /* Decirle al sistema que el modo input es RAW */
    control(CONSOLE, TC_MODER, 0, 0);

    /* inicializo adivina con guiones */
    for (i=0; i<80; i++)
            adivina[i] = '_';

    printf("(El juego no distingue mayusculas de
minusculas.)\n\r");
    printf("Ingrese una palabra y presione ENTER: ");
    i = 0;
    while (c != 13) {
            c = getchar();
            if (c == 13) {
                    secreta[i] = '\0'; /* fin de linea */
                    adivina[i] = '\0'; /* fin de linea */
                    break;
            }
            secreta[i] = c;
            i++;
    }
    n = i;

    /* borramos la linea anterior */
    printf("\r                                                    ");
    while(1) {
            printf("\rAdivine (vidas %d): %s\r", vidas,
adivina);
            ronda_ok = 0;
            c = getchar();
            for (i=0; i<n; i++) {
                    if ((secreta[i] == c) && (adivina[i] == '_'))
{
                            adivina[i] = c;
                            aciertos++;
                            ronda_ok = 1;
                    }
            }
            if (!ronda_ok) {
                    vidas--;
            }
            if (aciertos == n) {
                    printf("\n\rGANO!! \n\r");
                    break;
            }
            if (vidas == 0) {
                    printf("\n\rPerdio!! : \n\r");
                    break;
            }
    }
    printf("palabra secreta: %s\n\r", secreta);
    control(CONSOLE, TC_MODEC, 0, 0);
}
```

Cómo determinar si un año es un año bisiesto

Para determinar si un año es bisiesto, siga estos pasos:

1. Si el año es uniformemente divisible por 4, vaya al paso 2. De lo contrario, vaya al paso 5.
2. Si el año es uniformemente divisible por 100, vaya al paso 3. De lo contrario, vaya al paso 4.
3. Si el año es uniformemente divisible por 400, vaya al paso 4. De lo contrario, vaya al paso 5.
4. El año es un año bisiesto (tiene 366 días).
5. El año no es un año bisiesto (tiene 365 días).