Gabinete, Keith Ginoel S. 2020-03670
CMSC 125 – CD7L Dec. 04, 2023

ICS-OS Lab 03: Environment Variables, Processes, and Threads

PREREQUISITES:

To proceed with this lab, you should have completed Lab 02. Most of the commands that we will use in this lab will be run relative to the $ICSOS_HOME/ics-os directory. Update your local copy of the source code and create a new branch for this lab.

```
shio@ksgabinete:~$ ls
 Desktop     ics-os-kgsg    MOK.priv   Public      Videos
 Documents   IdeaProjects   Music      snap       'VirtualBox VMs'
 Downloads   MOK.der        Pictures   Templates   workspace
shio@ksgabinete:~$ cd ics-os-kgsg/
shio@ksgabinete:~/ics-os-kgsg$ cd ics-os/
shio@ksgabinete:~/ics-os-kgsg/ics-os$ checkout master
checkout: command not found
shio@ksgabinete:~/ics-os-kgsg/ics-os$ git checkout master
M       ics-os/base/icsos.hlp
M       ics-os/ics-os-floppy.img
M       ics-os/kernel/console/console.c
M       ics-os/kernel/dexapi/dex32API.c
M       ics-os/kernel/mapfile.txt
Already on 'master'
Your branch is up to date with 'origin/master'.
shio@ksgabinete:~/ics-os-kgsg/ics-os$ git branch
  lab01
  lab02
* master
shio@ksgabinete:~/ics-os-kgsg/ics-os$ git pull
Already up to date.
shio@ksgabinete:~/ics-os-kgsg/ics-os$ git checkout -b lab03
Switched to a new branch 'lab03'
shio@ksgabinete:~/ics-os-kgsg/ics-os$ git branch
  lab01
  lab02
* lab03
  master
```

TASKS:

Task 1: Environment Variables (3 points)

QUESTIONS:

1. What data structure is used for the implementation of environment variables?

```
EXPLORER
PROCESS
dex_taskmgr.c
environment.c 9+
environment.h
event.h
pdispatch.c
pdispatch.h
process.c
process.h
scheduler.c
scheduler.h
sync.c
sync.h
```

```c
/*
  Name: environment.c
  Copyright:
  Author: Joseph Emmanuel DL Dayo
  Date: 04/01/04 05:04
  Description: Handles environment strings with concurrency support
    As of the moment, implementation is on a global basis only, a per process
    implementation will be implemented in the future.
*/

#ifndef __ENVIRONMENT_H__
#define __ENVIRONMENT_H__


//A node in the environment variables data structure
//which is a doubly-linked list
typedef struct _env_strings {
    char *name;                 //name of the variable
    char *value;                //value of the variable
    struct _env_strings *next;  //pointer to the next node
    struct _env_strings *prev;  //pointer to the previous node
} env_strings;

int env_busywait = 0;           //used for synchronization
env_strings *env_head = 0;      //head of the list

/* function prototypes*/
void env_showenv();
char *env_getenv(const char *name, char *buf);
int env_setenv(const char *name, const char *value, int replace);
#endif
```

→ The environment variables in ICS-OS were implemented using the data structure Doubly-linked list.

2. Are environment variables unique for each process or shared by all processes?
→ In the screenshot provided for the implementation of the environment variables in ICS-OS, we can notice that there's a global variable named env_busywait. It is used for synchronization, like when we're working with multiple threads. With this, I'm assuming that the environment variables in ICS-OS are shared by all processes.

3. What are the functions used to set and get an environment variable?

```c
int env_setenv(const char *name, const char *value, int replace){
   env_strings *environment;

   //validate parameter
   if (strchr(name,'')!=0 || strchr(value,'')!=0)
      return -1;

   if (name==0)
      return -1;

   if (value==0)
      return -1;

   environment = env_getstring(name);

   //check if we can get in
   while (env_busywait)
      ;

   //we're in
   env_busywait = 1;
```

```c
   //environment does not exist yet
   if (environment == 0){
      int namelength=strlen(name), valuelength=strlen(value);

      environment = (env_strings*) malloc(sizeof(env_strings));

      //add to the beginning, insert at head
      environment->next = env_head;
      environment->prev = 0;
      if (env_head != 0)
        env_head->prev = environment;

      env_head = environment;

      //allocate spaces
      environment->name = (char*)malloc(namelength+1);
      environment->value =(char*)malloc(valuelength+1);

      //copy the strings
      strcpy(environment->name,name);
      strcpy(environment->value,value);
   else if (replace) { //replace the current value
      int valuelength=strlen(value);

      //resize the size of the value string
      environment->value=(char*)realloc(environment->value,valuelength+1);

      //update the value
      strcpy(environment->value,value);

   };
   //done!
   env_busywait = 0;
   return 0;
};
```

```c
//Return the value of an environment variable
char *env_getenv(const char *name, char *buf){

   //find the environment variable
   env_strings *ptr = env_getstring(name);

   //variable not found
   if (ptr == 0)
      return 0;

   //check if no one is accesing the environment variable
   while (env_busywait)
      ;

   //we're in, copy the value of the variable to buf
   env_busywait =1;
   strcpy(buf,ptr->value);

   //let go of the environment string
   env_busywait = 0;

   //return buf
   return buf;
};
```

→ The functions used for setting and getting an environment variable are env_setenv() and env_getenv() respectively.

4. Examine kernel/console/console.c. What console commands use the functions in question 3?

```c
if (strcmp(u,"cc") == 0){   //-- Builds a C program (invokes tcc.exe). Args: <name.exe> <name.c>
   char src[30],exe[30],cmdline[256],path[256];
   char sdk_home[128]="";
   env_getenv("SDK_HOME",sdk_home);
   env_getenv("PATH",path);
   if ( (strcmp(sdk_home,"")==0) || strcmp(path,"")==0 ){
      printf("Please set the SDK_HOME and PATH environment variables first.\n");
   }else{
      u=strtok(0," ");
      if (u!=0){
         strcpy(exe,u);
         u=strtok(0," ");
         if (u!=0){
            strcpy(src,u);
            sprintf(cmdline,"%s/tcc.exe -o%s %s -B%s %s/tccsdk.c %s/crt1.c",
                     path,exe,src,sdk_home,sdk_home,sdk_home);
            user_execp("/icsos/apps/tcc.exe",0,cmdline);
         }else{
            printf("Usage: cc <name.exe> <name.c>\n");
         }
      }else{
            printf("Usage: cc <name.exe> <name.c>\n");
      }
   }
}else
```

```c
if (strcmp(u,"set") == 0){    //-- Sets an environment variable. Args: <key>=<value>
   u=strtok(0," ");
   if (u==0){
      env_showenv();
   }else{
      char *name  = strtok(u,"=");
      char *value = strtok(0,"\n");
      env_setenv(name, value, 1);
   };
}else
```

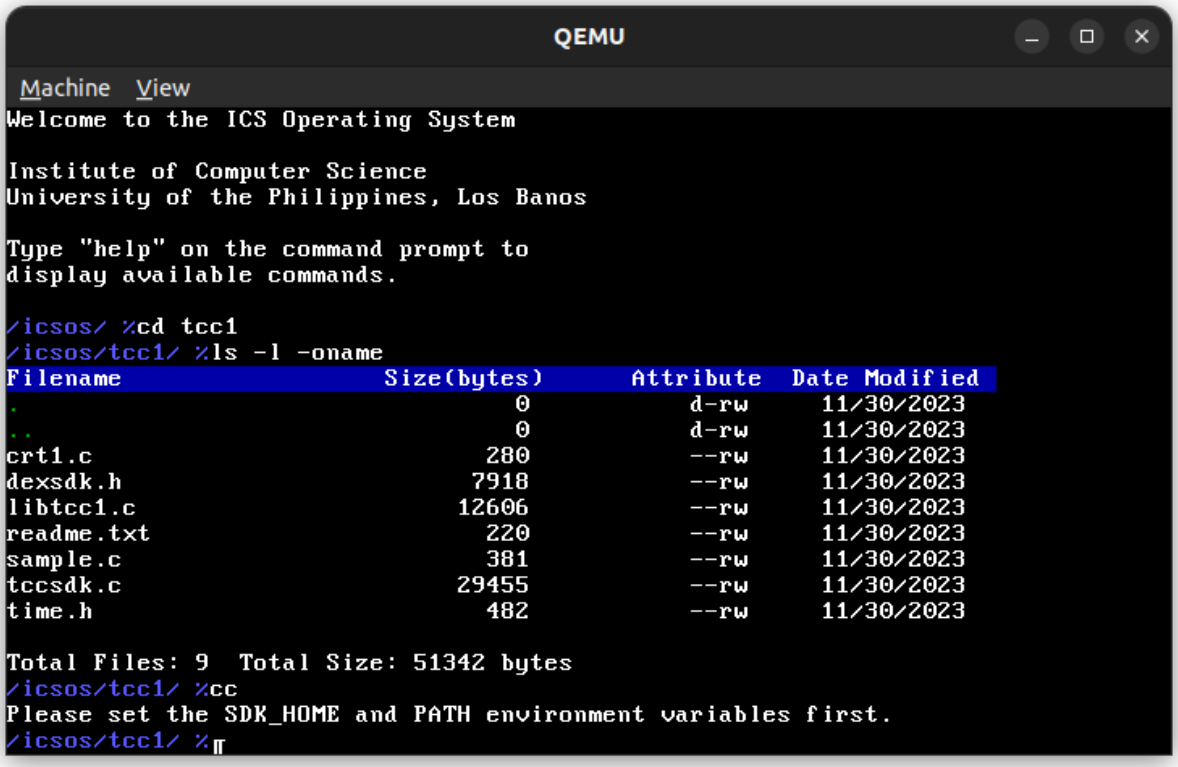
→ In ICS-OS, the console command 'cc' utilizes the function for getting an environment variable [env_getenv()] while the console command 'set' utilizes the function for setting an environment variable [env_setenv()].

Task 2: Edit and compile programs within ICS-OS (3 points)

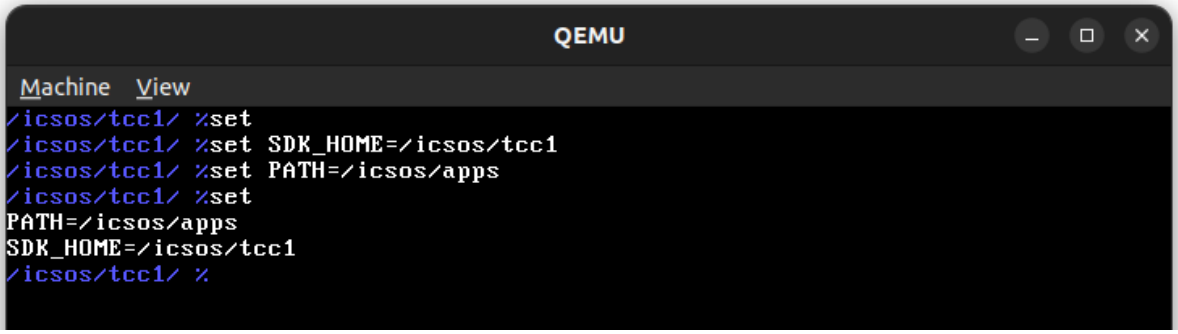
a.

Build and boot ICS-OS. Run the following command line sequence and capture a screen shot.

```
[1]% cd tcc1
[2]% ls -l -oname
[3]% cc
```



b. Setting environment variables:



c. Write and compile the envtest.c program.
Opening the ICS-OS simple editor:

```
QEMU
Machine  View
/icsos/tcc1/ %pwd
/icsos/tcc1/
/icsos/tcc1/ %ed.exe envtest.c
```

```
DexEdit - envtest.c(modified)                    col:18    row:8
int main(){
        char val[30];
        printf("Testing environment variables.\n");
        getenv("PATH",val);
        printf("PATH is %s\n",val);
        setenv("PERICO_HEART","liza s.");
        setenv("BETEL_HEART","enrique g.");
        return 0;_
}
```

Saving envtest.c:

```
Save file, Enter filename? [envtest.c] or "n":?
success!! {
        char val[30];
        printf("Testing environment variables.\n");
        getenv("PATH",val);
        printf("PATH is %s\n",val);
        setenv("PERICO_HEART","liza s.");
        setenv("BETEL_HEART","enrique g.");
        return 0;
}
```

Compiling:

```
/icsos/tcc1/ %ls
.                       ..                      crt1.c
dexsdk.h                envtest.c               libtcc1.c
readme.txt              sample.c                tccsdk.c
time.h
Total Files: 10  Total Size: 51554 bytes
/icsos/tcc1/ %cc envtest.exe envtest.c
tcc: file '/icso/tcc/crt1.c' not found
/icsos/tcc1/tccsdk.c:1385: warning: assignment makes pointer from integer withou
t a cast
/icsos/tcc1/tccsdk.c:1389: warning: assignment makes pointer from integer withou
t a cast
success!!
/icsos/tcc1/ %_
```

QUESTIONS:

1. Observe that you were able to run ed.exe in command line 'ed.exe envtest.c' without specifying its absolute path despite the current directory being /icsos/tcc1. Why is this so?

→ I was able to run ed.exe in command line ‘ed.exe envtest.c’ without specifying its absolute path despite the current directory being /icsos/tcc1 because of the environment variable PATH I set earlier. This environment variable has a value that specifies where in the filesystem executables can be found. Setting the environment variable PATH earlier with the value ‘/icsos/apps’ enables me to run an executable file located inside the specified directory even when I'm not in that particular directory.

2. Command line ‘envtest.exe’ will not work. Why? Show your fix to be able to run envtest.exe.
→ When we compile our code ‘envtest.c’ earlier with the console command ‘cc’, an executable file named ‘envtest.exe’ was created inside the /icsos/tcc1 directory. In order for ‘envtest.exe’ to work, we need to place it first inside the directory specified by the PATH variable. We can do this by using one of the ICS-OS internal commands called ‘copy’. This will allow us to duplicate a file from one directory to another.

Making a copy of ‘envtest.exe’ in ‘/icsos/apps’ directory:

```
/icsos/tcc1/ %copy /icsos/tcc1/envtest.exe /icsos/apps
copying to /icsos/apps/envtest.exe..
success!!
Copying 18496 bytes...
reading source file..
writing source file..
copy done.
```

Now we can enter the command ‘envtest.exe’ in the console again, and the following output will be displayed:

```
/icsos/tcc1/ %envtest.exe
Testing environment variables.
PATH is /icsos/apps
/icsos/tcc1/ %
```

3. After successfully running envtest.exe, what is the output of command line ‘set’?
Output of the command ‘set’ after running envtest.ext:

```
/icsos/tcc1/ %set
BETEL_HEART=enrique g.
PERICO_HEART=liza s.
PATH=/icsos/apps
SDK_HOME=/icsos/tcc1
/icsos/tcc1/ %
```

→ 2 new environment variables were added.

Does this support your answer in Q2 from Task 1?

Yes, this just proves my assumption in Q2 of Task 1 that environment variables are not unique for each process but shared by all processes. Take the executable files ed.exe and envtest.exe for example. They share the same environment variable PATH to be executed properly.

Task 3: Processes

Task 3.1: Process Control Block (4 points)

PCB implementation in ICS-OS (process.h):

```
process.h - Visual Studio Code
File Edit Selection View Go Run Terminal Help
Welcome | process.h | Extension: Docker | Extension: Dev Containers
home > shio > ics-os-kgsg > ics-os > kernel > process > process.h > _FPUregs > fpu
memory  Aa ab .* 4 of 7

/*Part of the PCB386 struct and is used for holding the registers of the
 FPU during context switching*/
typedef struct _FPUregs {
    DWORD fpu[27];
} FPUregs;

/*The primary data structure used for describing a process or thread,
  the first element of this data structure holds TSS data

  This is the process control block

*/
typedef struct _PCB386 {
    /*regs must be at the beginning of the PCB since the
     TSS directly points to the beginning of this structure*/
    saveregs regs;      /* TSS data, also for placing initial values for EAX,EBX etc.
                         A very hardware specific data structure for the Intel x86 family*/

    FPUregs regs2;      //stores the FPU registers, for Processors with FPU's

    DWORD size, version; //the size and version of the PCB386 structure is placed here
                          //this is for extensibility purposes

    DWORD *pagedirloc;  /*location of the processes active pagedirectory
                          NULL if page directory is the same as kernel's*/

    DWORD processid;    //a process ID that is generated by the system
```

QUESTIONS:

1. What field in the PCB describes the security bits for a process?

→ DWORD accesslevel;

```
    DWORD owner;        /*The owner refers to the pid of the process that spawned
                         this thread/process*/

    char name[256];     // the name of the process

    vfs_node *workdir;  //points to the vfs_node of the working directory

    DWORD accesslevel;  //the security bits for this process (kernel/system or user)

    DWORD priority;     //the process priority

    DWORD status;       //A flag for describing the status of the process
```

2. What field in the PCB describes the time the process arrived in the system?

→ DWORD arrivaltime;

```
    DWORD syscallsize;  //stores the size of the system call stack

    DWORD lastcputime, totalcputime; /*the taskswitcher increments this value every time
                                       the process takes control of the CPU*/

    DWORD arrivaltime;   //holds the time when this process arrived

    DWORD semhandle;    //holds the handle of the process semaphore

    process_mem *meminfo; /*points to a data structure containing the memory locatons taken up by
```

3. What field in the PCB describes the memory information used by a process?

→ process_mem *meminfo;

```
    DWORD arrivaltime;   //holds the time when this process arrived

    DWORD semhandle;    //holds the handle of the process semaphore

    process_mem *meminfo; /*points to a data structure containing the memory locatons taken up by
                            this process so that the process manager could clean this up easily.*/

    void *stackptr,*stackptr0; //user and system bottom of stack pointers
```

4. What field in the PCB describes the execution context(hardware specific) of a process?

→ saveregs regs;

```
*/
typedef struct _PCB386 {
    /*regs must be at the beginning of the PCB since the
     TSS directly points to the beginning of this structure*/
    saveregs regs;      /* TSS data, also for placing initial values for EAX,EBX etc.
                         A very hardware specific data structure for the Intel x86 family*/

    FPUregs regs2;      //stores the FPU registers, for Processors with FPU's

    DWORD size, version; //the size and version of the PCB386 structure is placed here
```

Task 3.2: Startup Processes (5 points)

Running the command ‘ps’:

```
/icsos/ %ps
dex32_scheduler  v1.00
Processes in memory:

PID    Name                Access Lvl  PPID              Size      AT          CT
[0  ] dex_kernel           kernel      0                  4K      0s     28s(20)%
[16 ] task_mgr       (t)   kernel      0                  4K     23s      5s(20)%
[17 ] disk_mgr       (t)   kernel      0                  4K     23s      5s(20)%
[18 ] fg_mgr         (t)   kernel      0                  4K     24s      5s(20)%
[19 ] console(0)     (t)   kernel      0                  4K     24s      5s(20)%

Total                  : 5 processes (20 KB)
Time Since Startup : 52
Legend: AT = Arrival Time, CT = CPU Time, %CT = Percent CPU Time
/icsos/ %
```

1. How many processes and kernel threads (those with (t) in the name) in total are running?
→ In total, there are 5 processes and kernel threads running. (4 of which are kernel threads)

2. What is the name of the process with PID 0?
→ The name of the process with PID 0 is dex_kernel;

3. What is the PID of console(0)? What is its access level?
→ The PID of console(0) is 19 and it has a kernel access level.

4. What function is used to create the running kernel threads?
The function createkthread in process.c is used to create the running kernel threads.

```c
/*
 * Creates a kernel thread. A kernel thread is owned and managed by the kernel.
 * It uses the address space of the kernel process.
 *
 *
 *
 */
DWORD createkthread(void *ptr,char *name,DWORD stacksize){

    PCB386 *temp=(PCB386*)malloc(sizeof(PCB386));          //allocate PCB for the thread
    DWORD cpuflags;

    memset(temp,0,sizeof(PCB386));                         //Initialize the PCB
    temp->before=current_process;                          //Point the 'before' to the parent
    strcpy(temp->name,name);                               //Set the name of the thread
    totalprocesses++;                                      //Increment the total number of processes
    temp->size       = sizeof(PCB386);
    temp->processid  = nextprocessid++;                    //Set the thread id
    temp->accesslevel = ACCESS_SYS;                        //Set the access level to kernel. This is a kernel thread.
    temp->owner      = getprocessid();                     //The owner of this thread is the current process
    temp->status     |= PS_ATTB_THREAD;                    //Indicate that this PCB is for a thread
    temp->knext      = kmext;                              //Set the the top of the heap of this thread to same as kernel
    temp->pagedirloc = pagedir1;                           //Set the pagedir for this thread to the first page directory
    temp->workdir    = current_process->workdir;           //Set the working directory of this thread to same as owner

    //set up the initial values of the CPU registers for this thread
    memset(temp.0,sizeof(saveregs));                       //This works because "regs" is the first field in the structure
    temp->regs.EIP   = (DWORD)ptr;                         //Set the function to be executed by this thread
    temp->stackptr   = malloc(stacksize);                    //Set up the stack for this thread
    temp->regs.ESP   = (DWORD)(temp->stackptr+stacksize-4);
    temp->stackptr   = (void*)temp->regs.ESP;
    temp->regs.CR3   = (DWORD)pagedir1;                    //Use the same memory as the kernel
    temp->regs.ES    = SYS_DATA_SEL;                       //Set the segment registers to appropriate selectors for kernel memory area.
    temp->regs.SS    = SYS_STACK_SEL;                      //this basically tells us that this is a kernel thread
    temp->regs.CS    = SYS_CODE_SEL;
    temp->regs.DS    = SYS_DATA_SEL;
    temp->regs.FS    = SYS_DATA_SEL;
    temp->regs.GS    = SYS_DATA_SEL;
    temp->regs.EFLAGS = 0x200;

    temp->arrivaltime = getprecisetime();                  //Store the arrival/creation time of this thread
    temp->stdin      = current_process->stdin;             //set the stdin

    /*Try to enter the critical section */
    sync_entercrit(&processmgr_busy);                      //Access to the process list must synched that's why its in the critical section
    dex32_stopints(&cpuflags);

    //add to the process list
    ps_enqueue(temp);                                      //Add the thread to the process list for scheduling

    //Exit the critical section
    dex32_restoreints(cpuflags);
    sync_leavecrit(&processmgr_busy);

    return temp->processid;
};
```

Some of the createkthread function calls (in kernel32.c):

```c
//Initialize the task manager - a module program that monitors processes
//for the user's convenience, as kernel thread
printf("Initializing the task manager...");
tm_pid=createkthread((void*)dex32_tm_updateinfo,"task_mgr",3500);

printf("[OK]\n");
```

```c
//create the IO manager thread which handles all I/O to and from
//block devices like the hard disk, floppy, CD-ROM etc. see iosched.c
printf("Initializing the disk manager...");
createkthread((void*)iomgr_diskmgr,"disk_mgr",200000);
printf("[OK]\n");


//Install a null block device
printf("Initializing the null block device...");
devfs_initnull();
```

```c
//create the foreground manager
fg_pid = createkthread((void*)fg_updateinfo,"fg_mgr",20000);

if (baremode)
    console_first++;
printf("dex32_startup(): Running console thread\n");

//Create a new console instance
consolepid = console_new();
```

5. To what function is the EIP register assigned to in the PCB of the very first process?

The EIP register is assigned to the function dex_init (in kernel32.c) in the PCB of the very first process.

```c
/* This procedure gets called when the kernel boots up, it sets up
   the initial processes that would be run. Use the ps command to
   view the details of these processes. Hardcoded. EIP field of the
   PCB points to the function that will be executed in the process.
*/
void process_init(){
    char tmp[255];
    PCB386 *kernel;


    //initialize the FPU
    asm volatile ("fininit");
    asm volatile ("fnsave ps_kernelfpustate");

    //Add the first process in memory which is the process kernel
    kernel=&kPCB;                                          //get a reference to the PCB
    memset(kernel,0,sizeof(PCB386));                       //initialize by zeroing it out
    kernel->next=kernel;                                   //next points to itself
    kernel->before=kernel;                                 //before points to itself
    kernel->processid=0;                                   //process id is 0
    kernel->meminfo=0;                                     //no memory information
    strcpy(kernel->name,"dex_kernel");                     //sPCB
    kernel->accesslevel=ACCESS_SYS;                        //kernel mode
    kernel->status = PS_ATTB_LOCKED | PS_ATTB_UNLOADABLE;  //it cannot be locked and unloaded, its the kernel!
    kernel->knext=kmext;                                   //top of the heap
    kernel->outdev=consoleDDL;                             //the console defined in kernel32.c, output of kernel goes here
    kernel->pagedirloc=pagedir1;                           //set page directory to the first location

    //initialize the current FPU state
    memcpy(&kernel->regs2,&ps_kernelfpustate,sizeof(ps_kernelfpustate));

    memset(&kernel->regs,0,sizeof(saveregs));              //initialize the execution context
    kernel->regs.EIP=(DWORD)dex_init;                      //dex_init() in kernel32.c
    kernel->regs.ESP= DISPATCHER_STACK_LOC;                //set the values of the registers for kernel mode process selector
    kernel->regs.CR3=pagedir1;
    kernel->regs.ES=SYS_DATA_SEL;
    kernel->regs.SS=SYS_STACK_SEL;
    kernel->regs.CS=SYS_CODE_SEL;
    kernel->regs.DS=SYS_DATA_SEL;
    kernel->regs.FS=SYS_DATA_SEL;
```

Task 3.3: Consoles (2 points)

Creating a new console:

```
/icsos/ %newconsole
New console thread created.
/icsos/ %
```

QUESTIONS:

1. Using ps, what is the name and PID of the new console? What is the name and PID of its parent process? Is the new console a process or a thread?

```
/icsos/ %ps
dex32_scheduler  v1.00
Processes in memory:

PID    Name                Access Lvl  PPID              Size      AT          CT
[0  ] dex_kernel           kernel      0                  4K      0s    224s(18)%
[16 ] task_mgr       (t)   kernel      0                  4K     23s    200s(18)%
[17 ] disk_mgr       (t)   kernel      0                  4K     23s    200s(18)%
[18 ] fg_mgr         (t)   kernel      0                  4K     24s    200s(18)%
[19 ] console(0)     (t)   kernel      0                  4K     24s    200s(18)%
[20 ] console(1)     (t)   kernel      19                 4K    693s     67s( 6)%

Total                : 6 processes (24 KB)
Time Since Startup : 1030
Legend: AT = Arrival Time, CT = CPU Time, %CT = Percent CPU Time
/icsos/ %_
```

→ The name and PID of the new console are 'console(1)' and '20', respectively.
→ The name and PID of the new console's parent process are 'console(0)' and '19', respectively.
→ The new console is a kernel thread (since it does have '(t)' along its name).

2. Study the implementation of the newconsole command in the kernel/console/console.c. What function is used to create a new console?

```
if (strcmp(u,"newconsole") == 0){   //-- Creates a new console.
    //create a new console
    console_new();
    printf("New console thread created.\n");
}else
```

```
int console_new(){
    //create a new console
    char consolename[255];
    sprintf(consolename,"console(%d)", console_first);
    return createkthread((void*)console, consolename, 200000);
};
```

The function console_new() is used to create a new console.

Task 3.4: User Processes (2 points)

Editing and compiling ‘count.c’:

```
DexEdit - count.c                                        col:519   row:519
int main() {
        int i;
        while(1) {
                printf("%d\n", (i=(i+1)%10));
                sleep(10);
        }
}
```

```
/icsos/tcc1/ %ls
.                       ..                      count.c
crt1.c                  dexsdk.h                envtest.c
envtest.exe             libtcc1.c               readme.txt
sample.c                tccsdk.c                time.h
Total Files: 12  Total Size: 70133 bytes
/icsos/tcc1/ %cc count.exe count.c
tcc: file '/icso/tcc/crt1.c' not found
/icsos/tcc1/tccsdk.c:1385: warning: assignment makes pointer from integer withou
t a cast
/icsos/tcc1/tccsdk.c:1389: warning: assignment makes pointer from integer withou
t a cast
success!!
/icsos/tcc1/ %
```

Copying ‘count.exe’ to ‘/icsos/apps’:

```
/icsos/tcc1/ %copy count.exe /icsos/apps
copying to /icsos/apps/count.exe..
Copying 18368 bytes...
reading source file..
writing source file..
copy done.
/icsos/tcc1/ %
```

Running ‘count.exe’ on a new console:

```
/icsos/tcc1/ %set
PATH=/icsos/apps
SDK_HOME=/icsos/tcc1
/icsos/tcc1/ %newconsole
New console thread created.
/icsos/tcc1/ %_
```

```
Machine   View
3
4
5
6
7
8
9
0
1
2
3
4
5
6
7
8
9
0
1
2
3
4
5
6
_
```

QUESTIONS:

1. What is the PID of count.exe process? What is its access level? How much memory does it use? What is its parent process?

```
/icsos/ %ps
dex32_scheduler  v1.00
Processes in memory:

PID   Name                Access Lvl  PPID              Size     AT          CT
[0  ] dex_kernel          kernel      0                   4K     0s     41s(14)%
[16 ] task_mgr       (t)  kernel      0                   4K    23s     17s(14)%
[17 ] disk_mgr       (t)  kernel      0                   4K    23s     17s(14)%
[18 ] fg_mgr         (t)  kernel      0                   4K    24s     17s(14)%
[19 ] console(0)     (t)  kernel      0                   4K    24s     17s(14)%
[20 ] console(1)     (t)  kernel      19                  4K    50s     12s(14)%
[21 ] /icsos/apps/count.exe    user        20              224K    63s     11s(1
4)%

Total                : 7 processes (248 KB)
Time Since Startup : 137
Legend: AT = Arrival Time, CT = CPU Time, %CT = Percent CPU Time
/icsos/ %_
```

→ The PID of count.exe process is 21 and it has a user access level.
→ count.exe uses 224k of memory.
→ Its parent process is the newly created console, 'console(1)', with PID 20.

Terminating count.exe:

```
QEMU
Machine  View
/icsos/tcc1/ %kill 21
/icsos/tcc1/ %ps
dex32_scheduler  v1.00
Processes in memory:

PID   Name                Access Lvl  PPID              Size     AT          CT
[0  ] dex_kernel          kernel      0                   4K     0s     35s(16)%
[16 ] task_mgr       (t)  kernel      0                   4K    23s     11s(16)%
[17 ] disk_mgr       (t)  kernel      0                   4K    23s     11s(16)%
[18 ] fg_mgr         (t)  kernel      0                   4K    24s     11s(16)%
[19 ] console(0)     (t)  kernel      0                   4K    24s     11s(16)%
[20 ] console(1)     (t)  kernel      19                  4K    58s      5s(16)%

Total                : 6 processes (24 KB)
Time Since Startup : 98
Legend: AT = Arrival Time, CT = CPU Time, %CT = Percent CPU Time
/icsos/tcc1/ %_
```

Task 3.5: Process Creation (5 points)

Editing and compiling meshell.c:

```
DexEdit - meshell.c(modified)                                   col:1     row:18
int main() {
        char cmdline[255]="";
        char params[255];
        char *cmd;
        printf("MeShell v1.0\n");
        printf("Type 'exit' to end session.\n");
        while (strcmp(cmdline,"exit")!=0) {
                printf("$");
                gets(cmdline);
                strcpy(params,cmdline);
                cmd=strtok(cmdline," ");
                if (cmd!=0){
                        if (!execp(cmd, 0, params))
                                printf("Executable not found.\n");
                }
        }
        return 0;

}
```

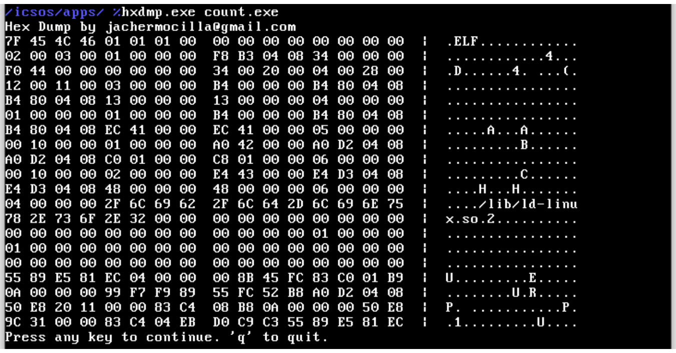
```
/icsos/tcc1/ %cc meshell.exe meshell.c
Please set the SDK_HOME and PATH environment variables first.
/icsos/tcc1/ %set SDK_HOME=/icsos/tcc1
/icsos/tcc1/ %set PATH=/icsos/apps
/icsos/tcc1/ %cc meshell.exe meshell.c
tcc: file '/icso/tcc/crt1.c' not found
meshell.c:11: warning: assignment makes pointer from integer without a cast
/icsos/tcc1/tccsdk.c:1385: warning: assignment makes pointer from integer withou
t a cast
/icsos/tcc1/tccsdk.c:1389: warning: assignment makes pointer from integer withou
t a cast
success!!
/icsos/tcc1/ %copy meshell.exe /icsos/apps
copying to /icsos/apps/meshell.exe..
success!!
Copying 18624 bytes...
reading source file..
writing source file..
copy done.
/icsos/tcc1/ %_
```

Running meshell.exe

```
/icsos/tcc1/ %meshell.exe
MeShell v1.0
Type 'exit' to end session.
$help
Executable not found.
$gg
Executable not found.
$exit
Executable not found.
/icsos/tcc1/ %
```

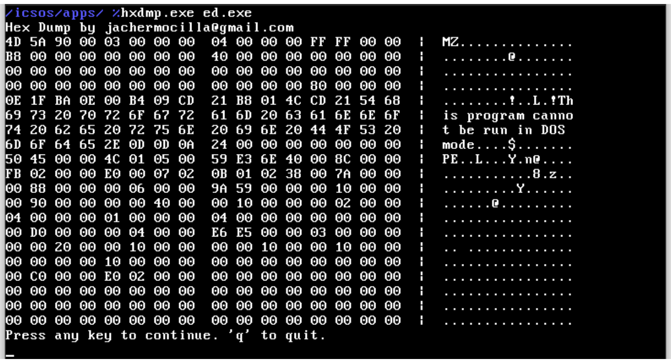
QUESTIONS

1. Use hxdmp.exe to determine the format of some of the executables in the apps folder. If the first few bytes has MZ then it is a windows executable, if ELF then it is a linux executable. What is the executable format of count.exe? ed.exe? tcc.exe? nasm.exe? meshell.exe?



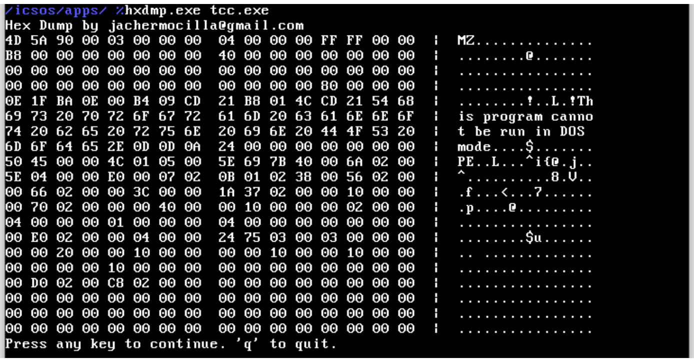
```
/icsos/apps/ %hxdmp.exe count.exe
Hex Dump by jachermocilla@gmail.com
7F 45 4C 46 01 01 01 00  00 00 00 00 00 00 00 00  ¦  .ELF............
02 00 03 00 01 00 00 00  F8 B3 04 08 34 00 00 00  ¦  ............4...
F0 44 00 00 00 00 00 00  34 00 20 00 04 00 28 00  ¦  .D......4. ...(.
12 00 11 00 03 00 00 00  B4 00 00 00 B4 80 04 08  ¦  ................
B4 80 04 08 13 00 00 00  13 00 00 00 04 00 00 00  ¦  ................
01 00 00 00 01 00 00 00  B4 00 00 00 B4 80 04 08  ¦  ................
B4 80 04 08 EC 41 00 00  EC 41 00 00 05 00 00 00  ¦  .....A...A......
00 10 00 00 01 00 00 00  A0 42 00 00 A0 D2 04 08  ¦  .........B......
A0 D2 04 08 C0 01 00 00  C8 01 00 00 06 00 00 00  ¦  ................
00 10 00 00 02 00 00 00  E4 43 00 00 E4 D3 04 08  ¦  .........C......
E4 D3 04 08 48 00 00 00  48 00 00 00 06 00 00 00  ¦  ....H...H.......
04 00 00 00 2F 6C 69 62  2F 6C 64 2D 6C 69 6E 75  ¦  ..../lib/ld-linu
78 2E 73 6F 2E 32 00 00  00 00 00 00 00 00 00 00  ¦  x.so.2..........
00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  ¦  ................
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
55 89 E5 81 EC 04 00 00  00 8B 45 FC 83 C0 01 B9  ¦  U.........E.....
0A 00 00 00 99 F7 F9 89  55 FC 52 B8 A0 D2 04 08  ¦  ........U.R.....
50 E8 20 11 00 00 83 C4  08 B8 0A 00 00 00 50 E8  ¦  P. ...........P.
9C 31 00 00 83 C4 04 EB  D0 C9 C3 55 89 E5 81 EC  ¦  .1.........U....
Press any key to continue. 'q' to quit.
```

→ count.exe is a linux executable



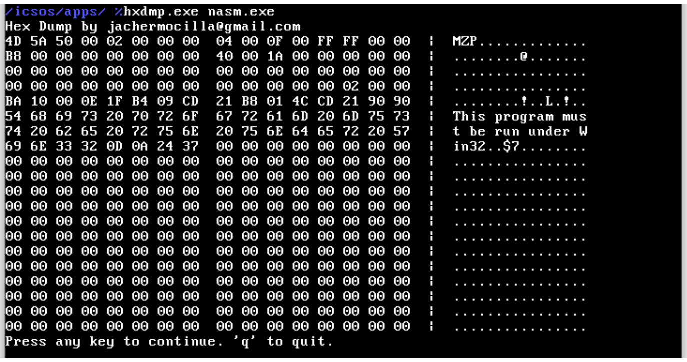
```
/icsos/apps/ %hxdmp.exe ed.exe
Hex Dump by jachermocilla@gmail.com
4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  ¦  MZ..............
B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ¦  ........@.......
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  ¦  ................
0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ¦  ........!..L.!Th
69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  ¦  is program canno
74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  ¦  t be run in DOS 
6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  ¦  mode....$.......
50 45 00 00 4C 01 05 00  59 E3 6E 40 00 8C 00 00  ¦  PE..L...Y.n@....
FB 02 00 00 E0 00 07 02  0B 01 02 38 00 7A 00 00  ¦  ...........8.z..
00 88 00 00 00 06 00 00  9A 59 00 00 00 10 00 00  ¦  .........Y......
00 90 00 00 00 00 40 00  00 10 00 00 00 02 00 00  ¦  ......@.........
04 00 00 00 01 00 00 00  04 00 00 00 00 00 00 00  ¦  ................
00 D0 00 00 00 04 00 00  E6 E5 00 00 03 00 00 00  ¦  ................
00 00 20 00 00 10 00 00  00 00 10 00 00 10 00 00  ¦  .. .............
00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 C0 00 00 E0 02 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
Press any key to continue. 'q' to quit.
_
```

→ ed.exe is a windows executable



```
/icsos/apps/ %hxdmp.exe tcc.exe
Hex Dump by jachermocilla@gmail.com
4D 5A 90 00 03 00 00 00  04 00 00 00 FF FF 00 00  ¦  MZ..............
B8 00 00 00 00 00 00 00  40 00 00 00 00 00 00 00  ¦  ........@.......
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 80 00 00 00  ¦  ................
0E 1F BA 0E 00 B4 09 CD  21 B8 01 4C CD 21 54 68  ¦  ........!..L.!Th
69 73 20 70 72 6F 67 72  61 6D 20 63 61 6E 6E 6F  ¦  is program canno
74 20 62 65 20 72 75 6E  20 69 6E 20 44 4F 53 20  ¦  t be run in DOS 
6D 6F 64 65 2E 0D 0D 0A  24 00 00 00 00 00 00 00  ¦  mode....$.......
50 45 00 00 4C 01 05 00  5E 69 7B 40 00 6A 02 00  ¦  PE..L...^i{@.j..
5E 04 00 00 E0 00 07 02  0B 01 02 38 00 56 02 00  ¦  ^..........8.V..
00 66 02 00 00 3C 00 00  1A 37 02 00 00 10 00 00  ¦  .f...<...7......
00 70 02 00 00 00 40 00  00 10 00 00 00 02 00 00  ¦  .p....@.........
04 00 00 00 01 00 00 00  04 00 00 00 00 00 00 00  ¦  ................
00 E0 02 00 00 04 00 00  24 75 03 00 03 00 00 00  ¦  ........$u......
00 00 20 00 00 10 00 00  00 00 10 00 00 10 00 00  ¦  .. .............
00 00 00 00 10 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 D0 02 00 C8 02 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
Press any key to continue. 'q' to quit.
```

→ tcc.exe is a windows executable



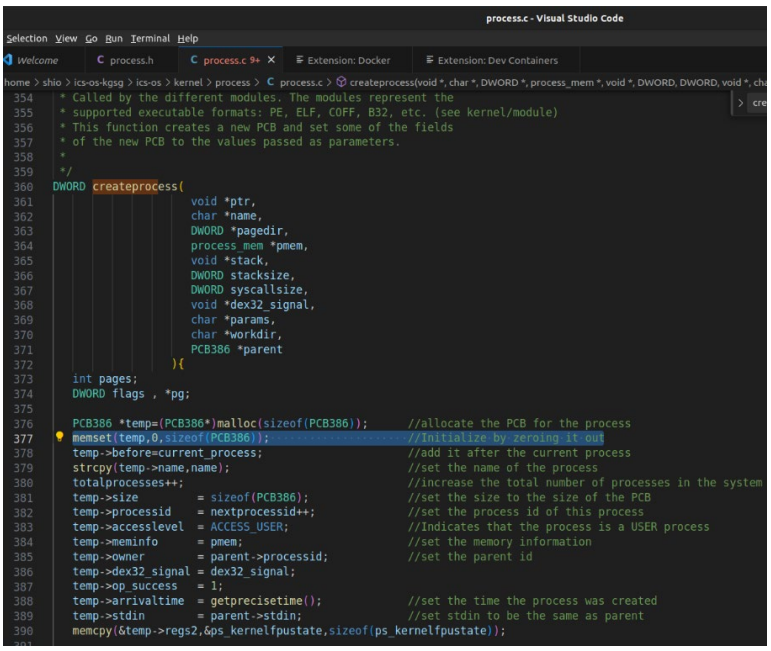
```
/icsos/apps/ %hxdmp.exe nasm.exe
Hex Dump by jachermocilla@gmail.com
4D 5A 50 00 02 00 00 00  04 00 0F 00 FF FF 00 00  ¦  MZP.............
B8 00 00 00 00 00 00 00  40 00 1A 00 00 00 00 00  ¦  ........@.......
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 02 00 00  ¦  ................
BA 10 00 0E 1F B4 09 CD  21 B8 01 4C CD 21 90 90  ¦  ........!..L.!..
54 68 69 73 20 70 72 6F  67 72 61 6D 20 6D 75 73  ¦  This program mus
74 20 62 65 20 72 75 6E  20 75 6E 64 65 72 20 57  ¦  t be run under W
69 6E 33 32 0D 0A 24 37  00 00 00 00 00 00 00 00  ¦  in32..$7........
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
Press any key to continue. 'q' to quit.
```

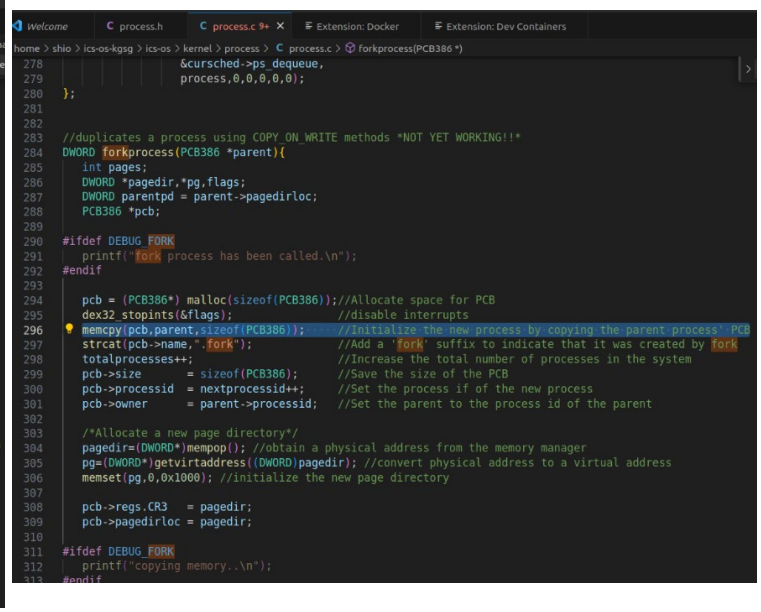
→ nasm.exe is a windows executable

```
/icsos/apps/ %hxdmp.exe meshell.exe
Hex Dump by jachermocilla@gmail.com
7F 45 4C 46 01 01 01 00  00 00 00 00 00 00 00 00  ¦  .ELF............
02 00 03 00 01 00 00 00  CB B4 04 08 34 00 00 00  ¦  ............4...
F0 45 00 00 00 00 00 00  34 00 20 00 04 00 28 00  ¦  .E......4. ...(.
12 00 11 00 03 00 00 00  B4 00 00 00 B4 80 04 08  ¦  ................
B4 80 04 08 13 00 00 00  13 00 00 00 04 00 00 00  ¦  ................
01 00 00 00 01 00 00 00  B4 00 00 00 B4 80 04 08  ¦  ................
B4 80 04 08 AC 42 00 00  AC 42 00 00 05 00 00 00  ¦  .....B...B......
00 10 00 00 01 00 00 00  60 43 00 00 60 D3 04 08  ¦  ........`C..`...
60 D3 04 08 00 02 00 00  08 02 00 00 06 00 00 00  ¦  `...............
00 10 00 00 02 00 00 00  EC 44 00 00 EC D4 04 08  ¦  .........D......
EC D4 04 08 48 00 00 00  48 00 00 00 06 00 00 00  ¦  ....H...H.......
04 00 00 00 2F 6C 69 62  2F 6C 64 2D 6C 69 6E 75  ¦  ..../lib/ld-linu
78 2E 73 6F 2E 32 00 00  00 00 00 00 00 00 00 00  ¦  x.so.2..........
00 00 00 00 00 00 00 00  00 00 00 00 01 00 00 00  ¦  ................
01 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  ¦  ................
55 89 E5 81 EC 04 02 00  00 B8 00 00 00 00 88 85  ¦  U...............
01 FF FF FF B8 FE 00 00  00 50 B8 00 00 00 00 50  ¦  .........P.....P
8D 85 02 FF FF FF 50 E8  EE 1F 00 00 83 C4 0C B8  ¦  ......P.........
60 D3 04 08 50 E8 DF 11  00 00 83 C4 04 B8 6E D3  ¦  `...P........n.
Press any key to continue. 'q' to quit.
```

→ meshell.exe is a linux executable

2. Which line in the forkprocess() and createprocess() functions initializes the PCB of the new process. How do the functions differ?





→ In the process.c file, following the allocation of memory space for the PCB using the malloc function, the createprocess() and forkprocess() functions initialize the PCB of the new process at lines 377 and 296, respectively. In the createprocess function, the memset() function is invoked to initialize the value of the allocated memory space to zero. In contrast, the memcpy() function is utilized in the forkprocess function to initialize the value of the allocated memory space with a pre-existing PCB.

→ The distinction between the two functions (createprocess() and forkprocess()) is that the createprocess() function creates the PCB of a new process from scratch, while the forkprocess function duplicates the PCB of the new process's parent process.

Editing and compiling fork.c:

```
DexEdit - fork.c                                        col:25    row:21
int NITER=10000;
int main() {
        int retval, i;
        printf("Calling fork()..\n");
        retval = fork();

        if (retval ==0) {
                printf("In child. mypid:%d\n",getpid());
                for (i=0;i<NITER;i++) {
                        printf("Child: %d\n",i);
                        sleep(10);
                }
                exit(0);
        }
        else if (retval > 0) {
                printf("In parent, mypid:%d, child pid: %d\n", getpid(), retval)
                for (i=0;i<NITER;i++) {
                        printf("Parent: %d\n", i);
                        sleep(15);
                }
                exit(0);_
        }
        return -1;
}
```

```
/icsos/tcc1/ %ls
.                       ..                      count.c
count.exe               crt1.c                  dexsdk.h
envtest.c               envtest.exe             fork.c
libtcc1.c               meshell.c               meshell.exe
readme.txt              sample.c                tccsdk.c
time.h
Total Files: 16  Total Size: 107929 bytes
/icsos/tcc1/ %cc fork.exe fork.c
tcc: file '/icso/tcc/crt1.c' not found
/icsos/tcc1/tccsdk.c:1385: warning: assignment makes pointer from integer withou
t a cast
/icsos/tcc1/tccsdk.c:1389: warning: assignment makes pointer from integer withou
t a cast
success!!
/icsos/tcc1/ %
```

Creating a new console to run meshell.exe:

```
/icsos/ %cd apps
/icsos/apps/ %newconsole
New console thread created.
```

Running meshell.exe on the new console:

```
/icsos/apps/ %meshell.exe
MeShell v1.0
Type 'exit' to end session.
```

Running fork.exe in meshell:

```
/icsos/apps/ %meshell.exe
MeShell v1.0
Type 'exit' to end session.
$fork.exe
Calling fork()..
In child. mypid:23
Child: 0
In parent, mypid:22, child pid: 23
Parent: 0
Child: 1
Parent: 1
Child: 2
Parent: 2
Child: 3
Child: 4
Parent: 3
Child: 5
Parent: 4
Child: 6
Parent: 5
Child: 7
Child: 8
Parent: 6
Child: 9
```

Updated list of processes:

```
QEMU
Machine  View
/icsos/ %cd apps
/icsos/apps/ %newconsole
New console thread created.
/icsos/apps/ %ps
dex32_scheduler  v1.00
Processes in memory:

PID   Name                Access Lvl  PPID              Size     AT          CT
[0  ] dex_kernel          kernel      0                  4K     0s     42s(31)%
[16 ] task_mgr       (t)  kernel      0                  4K    23s     18s(13)%
[17 ] disk_mgr       (t)  kernel      0                  4K    23s     18s(13)%
[18 ] fg_mgr         (t)  kernel      0                  4K    24s     18s(13)%
[19 ] console(0)     (t)  kernel      0                  4K    24s     18s(13)%
[20 ] console(1)     (t)  kernel      19                 4K    68s     10s( 7)%
[21 ] /icsos/apps/meshell.exe     user        20                 224K   107s      3s
( 2)%
[22 ] fork.exe            user        21                224K   124s      2s( 1)%
[23 ] fork.exe.fork       user        22                444K   124s      2s( 1)%

Total                : 9 processes (916 KB)
Time Since Startup : 142
Legend: AT = Arrival Time, CT = CPU Time, %CT = Percent CPU Time
/icsos/apps/ %
```

Task 3.6: Process Termination (2 points)
QUESTIONS:

1. What function is called to kill a kernel process/thread?
→ The dex32_killkthread() function is the one being called to kill a kernel process/thread.

```
//used to kill kernel (Ring0) threads only!!!
DWORD dex32_killkthread(DWORD processid){
   PCB386 *ptr;
   PCB386 *end;
   DWORD flags;

   sync_entercrit(&processmgr_busy);

   //get a pointer to the PCB
   ptr = bridges_ps_findprocess(processid);

   //stop interrupts
   dex32_stopints(&flags);

   if (ptr != -1){
      if (!ptr->status & PS_ATTB_UNLOADABLE){

         PCB386 *parent;

         //kernel thread
         if (ptr->accesslevel == ACCESS_SYS)
            free(ptr->stackptr);

         if (ptr->semhandle != 0)
            set_semaphore(ptr->semhandle,SIG_TERM);

         ps_dequeue(ptr);

         free(ptr);
         dex32_restoreints(flags);

         sync_leavecrit(&processmgr_busy);

         return 1;

      };

   };

   dex32_restoreints(flags);
   sync_leavecrit(&processmgr_busy);
   return 0;
}
```

2. What function is called to kill a user thread?
→ The kill_thread() function is called to kill a user thread.

```
//Kill user threads
DWORD kill_thread(PCB386 *ptr){
   DWORD flags;
   dex32_stopints(&flags);

   kill_children(ptr->processid); //kill the children of this thread first!!cascade kill

   //Tell the scheduler to remove this thread from the ready queue
   ps_dequeue(ptr);

   if (ptr->stackptr0!=0)
         free(ptr->stackptr0);

   free(ptr);
   dex32_restoreints(flags);
   return 1;
   ;
};

//Iterates over the process list to terminate children processes
DWORD kill_children(DWORD processid){
   PCB386 *ptr;
   sync_entercrit(&processmgr_busy);

   ptr = bridges_ps_findprocess(processid);

   if (ptr!=-1){

      if (ptr->owner==processid && !( ptr->status&PS_ATTB_UNLOADABLE ) ){
         //kill_thread(ptr);
         kill_process(ptr->processid);
         sync_leavecrit(&processmgr_busy);
         return 1;
      };
   };

   sync_leavecrit(&processmgr_busy);
   return 0;
};
```

Reflection
Write some realizations and questions that crossed your mind while doing this lab.

Doing this lab exercise made me realize how crucial APIs really are in big projects like ICS-OS. The well-designed ICS-OS API helped me understand how different parts of the OS behave and communicate with each other, making it easier for me to determine which sections of the code one should modify to add some new feature implementation without worrying too much about messing up the entire behavior of the OS. This exercise also reminded me of just how little knowledge I have about operating systems in general. Before, I didn't even know the purpose of the PATH environment variable in my system. Despite encountering it multiple times, especially when installing programming languages like Python or Java, I never clearly understood why including them in the PATH was mandatory. Thankfully, after completing this lab, I've come to learn the true purpose of the PATH variable and, in general, the significance of environment variables in an operating system like ICS-OS.

The only two questions that come to mind during this lab exercise are: "Can a process with kernel access-level be forked?" and "Why can't I kill a console process in ICS-OS?" I just noticed that when I execute fork.exe directly inside a console, its access-level is limited to users only. Shouldn't it be supposed to be at the kernel access-level? Since it's just basically making a duplicate of its parent process - which is, in fact, a kernel process? Or is there an underlying implementation of the code somewhere that I didn't know of that limits a forked process to have user access-level only? Now, for the second question, despite multiple attempts, I still can't find success in killing a console process. Is it really not possible to terminate a console process in ICS-OS, or is there another command specifically designed for that purpose?