컴퓨터학부 20142385 엄재식

1. 과제 개요
2. 키보드를 눌렀을 때와 뗄 때 스캔코드가 발생한다. 이를 이용해 Ctrl+l, Ctrl+tab 조합키를 구현
3. Ctrl+l 입력 시 커서가 있는 라인이 콘솔 상에서 가장 위의 라인이 되도록 스크롤 구현
4. Ctrl+tab 입력 시 foreground 프로세스를 다음 프로세스로 바꾼다. 프로세스의 입출력버퍼도 함께 바뀐다.
5. 상세설계
6. kbd.c

원래 코드에서 사용하던 전역변수인 kbd\_buf와 buf\_head, buf\_tail을 cur\_foreground\_process->kbd\_buffer의 멤버변수로 바꾸어 줌

init\_kbd 함수에서 Ctrl을 눌러야 하는 조합키를 구현하기 위해 Ctrl\_flag를 추가하여 초기화해줌, 전역 키보드 버퍼를 초기화하던 것을 프로세스가 사용할 kbd\_buffer를 초기화 하도록 바꾸어 줌

UpdateKeyStat 함수에서 스캔코드를 읽어 Ctrl\_flag를 바꾸도록 코드추가

ConvertScancodeToASCII 함수에서 Ctrl+l을 입력해줄 때 clearScreen 함수를 호출하여 콘솔클리어를 함,

Ctrl+tab을 입력해줄 때 next\_foreground\_proc 함수를 호출하여 다음 foreground프로세스로 바꿈

kbd\_read\_char 함수에서 입력버퍼를 구분하기 위해 프로세스를 구분하여 동작을 수행하도록 제어문 추가

1. console.c

원래 코드에서 사용하던 전역변수인 buf\_s, buf\_w, buf\_p, sum\_y, a\_s을 cur\_console의 멤버변수로 바꾸어 줌

init\_console 함수에서 전역 콘솔을 초기화하던 것을 프로세스가 사용할 콘솔을 초기화 하도록 바꾸어 줌, login프로세스가 나오기전 출력할 콘솔을 하나 골라 cur\_console로 만들어주고 초기화함

clearScreen 함수에서 커서라인이 맨 위로 가도록 cur\_console의 glob\_y가 0이 될 때까지 scroll을 호출하도록 scroll\_screen을 참고하여 코드를 수정하여 콘솔 클리어 구현

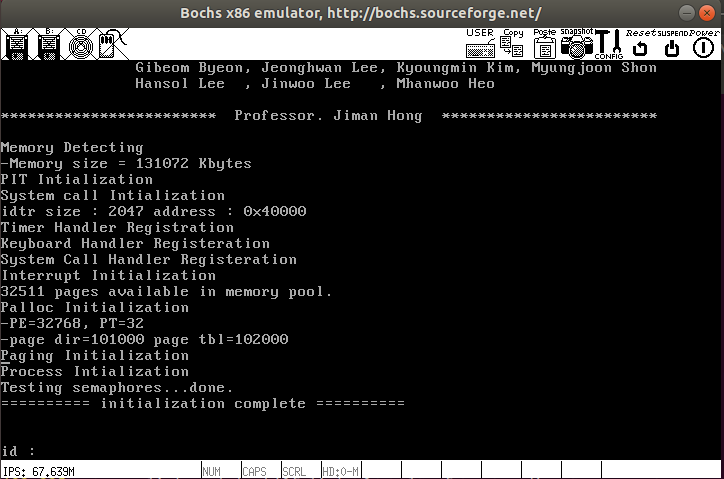
printk 함수에서 호출한 프로세스의 콘솔이 현재 사용하는 콘솔일 때 출력할 수 있도록 제어문을 추가함

1. proc.c

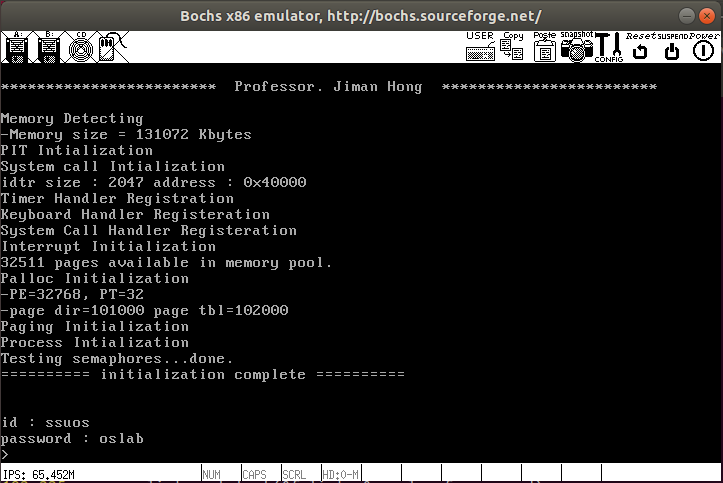
init\_proc 함수에서 처음 프로세스를 foreground 리스트에 추가되도록 f\_list에 추가해줌, 처음 사용할 프로세스에 콘솔과 키보드버퍼를 등록해줌

process\_create 함수에서 인자로 추가된 opt를 보고 foreground일 때 콘솔, 키보드 버퍼를 등록해주고 cur\_foreground\_process 갱신과 f\_list에 추가를 해 줌, background일 때 콘솔을 부모 프로세스의 콘솔로 등록해주고 키보드 버퍼는 등록하지 않도록 해 줌

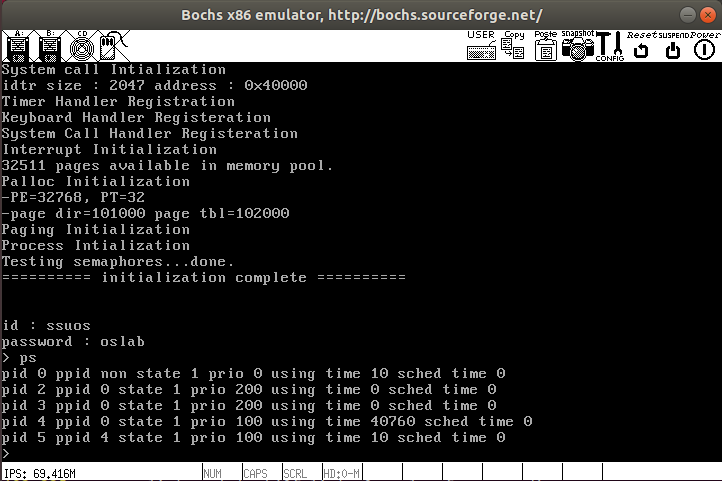
1. 실행결과

처음 부팅

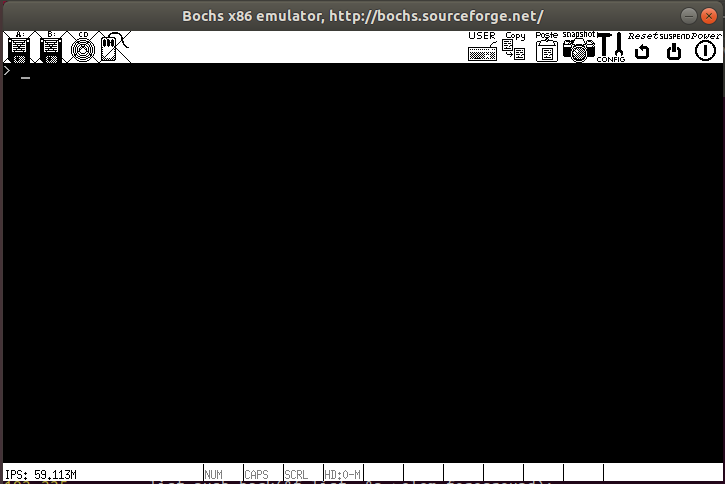
로그인



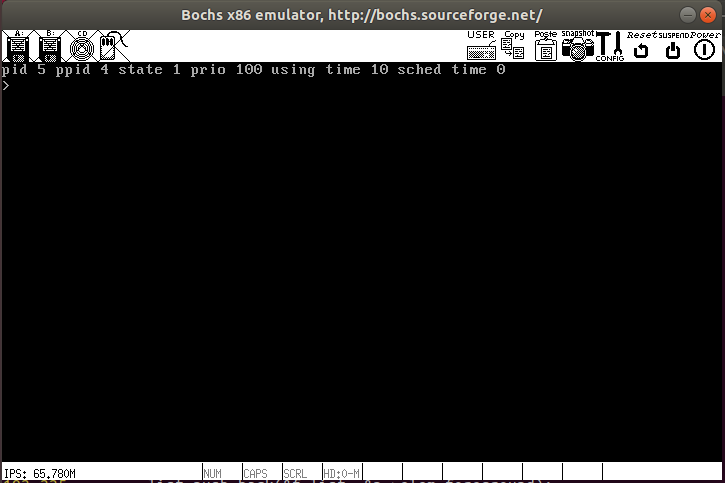
ps 출력



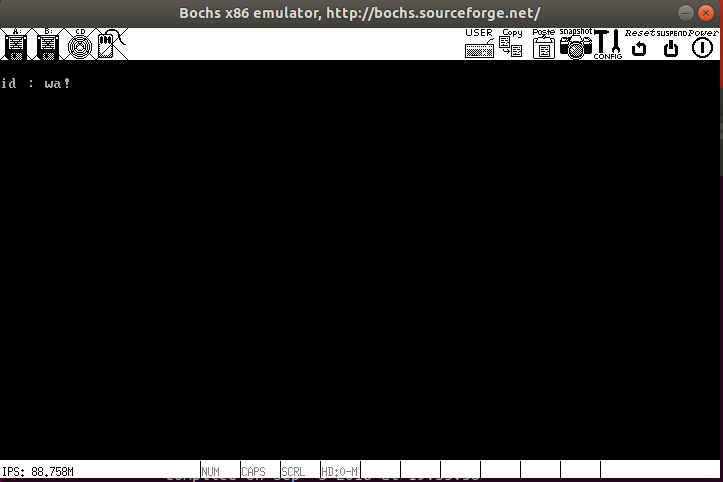
Ctrl+l 입력



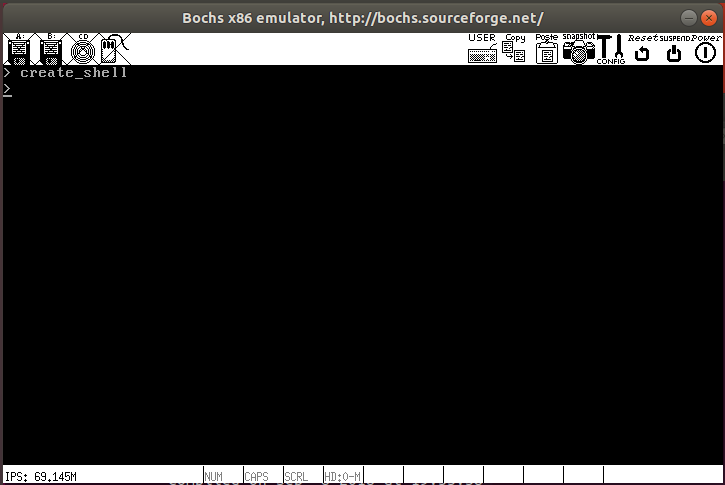
Page\_up 입력



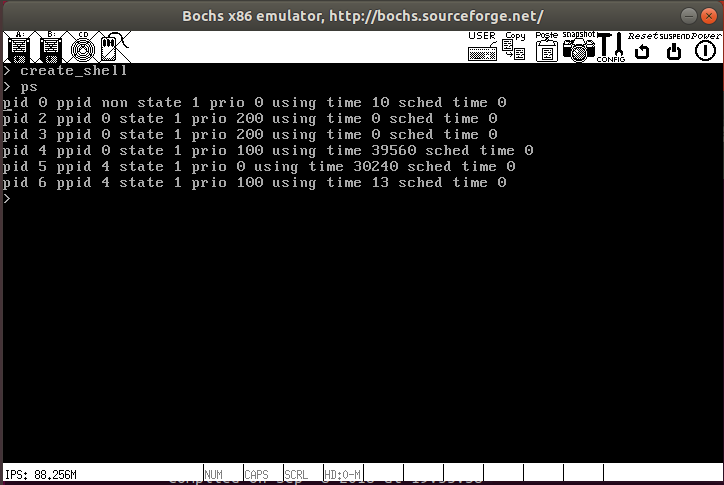
create\_shell 명령어입력



Ctrl+tab 입력



ps입력



1. 소스코드
2. kbd.c

#include <device/kbd.h>  
#include <type.h>  
#include <device/console.h>  
#include <interrupt.h>  
#include <device/io.h>  
#include <ssulib.h>  
#include <proc/proc.h>  
  
static Key\_Status KStat;  
  
//static char kbd\_buf[BUFSIZ];  
//int buf\_head, buf\_tail;  
//위 전역변수를 사용하는 코드를 cur\_foreground\_process를 사용하는 코드로 변경  
extern struct process \*cur\_foreground\_process;  
extern struct process \*cur\_process;  
Kbd\_buffer kbd\_buffer[MAX\_KBD\_BUFFER]={0};  
  
static BYTE Kbd\_Map[4][KBDMAPSIZE] = {  
    { /\* default \*/  
        0x00, 0x00, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', '\b', '\t',  
        'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']', '\n', 0x00, 'a', 's',  
        'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', '\'', '`', 0x00, '\\', 'z', 'x', 'c', 'v',  
        'b', 'n', 'm', ',', '.', '/', 0x00, 0x00, 0x00, ' ', 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, '-', 0x00, 0x00, 0x00, '+', 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00  
    },  
    { /\* capslock \*/  
        0x00, 0x00, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', '\b', '\t',  
        'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', '[', ']', '\n', 0x00, 'A', 'S',  
        'D', 'F', 'G', 'H', 'J', 'K', 'L', ';', '\'', '`', 0x00, '\\', 'Z', 'X', 'C', 'V',  
        'B', 'N', 'M', ',', '.', '/', 0x00, 0x00, 0x00, ' ', 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, '-', 0x00, 0x00, 0x00, '+', 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00  
    },  
    { /\* Shift \*/  
        0x00, 0x00, '!', '@', '#', '$', '%', '^', '&', '\*', '(', ')', '\_', '+', '\b', 0x00,  
        'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', '{', '}', 0x00, 0x00, 'A', 'S',  
        'D', 'F', 'G', 'H', 'J', 'K', 'L', ':', '\"', '~', 0x00, '|', 'Z', 'X', 'C', 'V',  
        'B', 'N', 'M', '<', '>', '?', 0x00, 0x00, 0x00, ' ', 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, '-', 0x00, 0x00, 0x00, '+', 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00  
    },  
    { /\* Capslock & Shift  \*/  
        0x00, 0x00, '!', '@', '#', '$', '%', '^', '&', '\*', '(', ')', '\_', '+', '\b', 0x00,  
        'q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '{', '}', 0x00, 0x00, 'a', 's',  
        'd', 'f', 'g', 'h', 'j', 'k', 'l', ':', '\"', '~', 0x00, '|', 'z', 'x', 'c', 'v',  
        'b', 'n', 'm', '<', '>', '?', 0x00, 0x00, 0x00, ' ', 0x00, 0x00, 0x00, 0x00, 0x00, 0x00,      
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, '-', 0x00, 0x00, 0x00, '+', 0x00,  
        0x00, 0x00, 0x00, 0x00, 0x00, 0x00  
    }  
};  
  
static bool kbd\_remove\_char();  
  
void init\_kbd(void)  
{  
    KStat.ShiftFlag = 0;  
    KStat.CapslockFlag = 0;  
    KStat.NumlockFLag = 0;  
    KStat.ScrolllockFlag = 0;  
    KStat.ExtentedFlag = 0;  
    KStat.PauseFlag = 0;  
    KStat.CtrlFlag = 0;//컨트롤 플래그 추가  
    //buf\_head = 0;  
    //buf\_tail = 0;  
    //키보드 버퍼 초기화  
    int i;  
    for(i=0;i<MAX\_KBD\_BUFFER;i++)  
    {  
        kbd\_buffer[i].head = 0;  
        kbd\_buffer[i].tail = 0;  
    }  
  
    reg\_handler(33, kbd\_handler);  
}  
  
void UpdateKeyStat(BYTE Scancode)  
{  
    if(Scancode & 0x80)  
    {  
        if(Scancode == 0xB6 || Scancode == 0xAA)  
        {  
            KStat.ShiftFlag = FALSE;  
        }  
          
        //컨트롤키 뗄때  
        if(Scancode == 0x9D)  
        {  
            KStat.CtrlFlag = FALSE;  
        }  
    }  
    else  
    {  
        if(Scancode == 0x3A && KStat.CapslockFlag)  
        {  
            KStat.CapslockFlag = FALSE;  
        }  
        else if(Scancode == 0x3A)  
            KStat.CapslockFlag = TRUE;  
        else if(Scancode == 0x36 || Scancode == 0x2A)  
        {  
            KStat.ShiftFlag = TRUE;  
        }  
        else if(Scancode == 0x1D)//컨트롤키 눌렀을 때  
        {  
            KStat.CtrlFlag = TRUE;  
        }  
  
    }  
  
    if(Scancode == 0xE0)  
    {  
        KStat.ExtentedFlag = TRUE;  
    }  
    else if(KStat.ExtentedFlag == TRUE && Scancode != 0xE0)  
    {  
        KStat.ExtentedFlag = FALSE;  
    }  
}  
  
BOOL ConvertScancodeToASCII(BYTE Scancode, BYTE \*Asciicode)  
{  
    if(KStat.PauseFlag > 0)  
    {  
        KStat.PauseFlag--;  
        return FALSE;  
    }  
  
    if(KStat.ExtentedFlag == TRUE)  
    {  
        if(Scancode & 0x80)      
            return FALSE;  
        \*Asciicode = Scancode;  
        return TRUE;  
    }  
  
    if(Scancode == 0xE1)  
    {  
        \*Asciicode = 0x00;  
        KStat.PauseFlag = 2;  
        return FALSE;  
    }  
    else if(Scancode == 0xE0)  
    {  
        \*Asciicode = 0x00;  
        return FALSE;  
    }  
  
    if(!(Scancode & 0x80))  
    {  
        if(KStat.ShiftFlag & KStat.CapslockFlag)  
        {  
            \*Asciicode = Kbd\_Map[3][Scancode & 0x7F];  
        }  
        else if(KStat.ShiftFlag)  
        {  
            \*Asciicode = Kbd\_Map[2][Scancode & 0x7F];  
        }  
        else if(KStat.CapslockFlag)  
        {  
            \*Asciicode = Kbd\_Map[1][Scancode & 0x7F];  
        }  
        else if(KStat.CtrlFlag)  
        {  
            \*Asciicode = 0x00;  
            if(Scancode == 0x26)//컨트롤 플래그 on 일 때 l키 입력  
                clearScreen();//콘솔 클리어  
                //printk("%s", "Ctrl+l\n");  
            else if(Scancode == 0x0f)//컨트롤 플래그 on 일 때 tab키 입력  
                next\_foreground\_proc();//cur\_foreground\_process 다음으로 교체  
                //printk("%s", "Ctrl+tab\n");  
            return FALSE;  
        }  
        else  
        {  
            \*Asciicode = Kbd\_Map[0][Scancode];  
        }  
  
        return TRUE;  
    }  
    return FALSE;  
}  
  
bool isFull()  
{  
    //return (buf\_head-1) % BUFSIZ == buf\_tail;  
    return (cur\_foreground\_process->kbd\_buffer->head - 1) % BUFSIZ == cur\_foreground\_process->kbd\_buffer->tail;  
}  
  
bool isEmpty()  
{  
    //return buf\_head == buf\_tail;  
    return cur\_foreground\_process->kbd\_buffer->head == cur\_foreground\_process->kbd\_buffer->tail;  
}  
  
void kbd\_handler(struct intr\_frame \*iframe)  
{  
    BYTE asciicode;  
    BYTE data = inb(0x60);  
  
    if(ConvertScancodeToASCII(data, &asciicode))  
    {  
  
#ifdef SCREEN\_SCROLL  
        if( KStat.ExtentedFlag == TRUE)  
        {  
            switch(asciicode)  
            {  
                case 72://UP  
                    break;  
                case 80 ://DOWN  
                    break;  
                case 75 ://LEFT  
                    break;  
                case 77 ://RIGHT  
                    break;  
                case 73 :// PageUp  
                    scroll\_screen(-1);  
                    break;  
                case 81 :// PageDown  
                    scroll\_screen(+1);  
                    break;  
                case 79 ://End  
                    set\_fallow();  
                    break;  
                case 0x1d:  
                    KStat.CtrlFlag = TRUE;  
                    break;  
                case 0x9d:  
                    KStat.CtrlFlag = FALSE;  
                    break;  
                default:  
                    break;  
            }  
        }  
        else if( !isFull() && asciicode != 0)  
        {  
            //kbd\_buf[buf\_tail] = asciicode;  
            //buf\_tail = (buf\_tail + 1) % BUFSIZ;  
            cur\_foreground\_process->kbd\_buffer->buf[cur\_foreground\_process->kbd\_buffer->tail] = asciicode;  
            cur\_foreground\_process->kbd\_buffer->tail = (cur\_foreground\_process->kbd\_buffer->tail + 1) % BUFSIZ;  
  
        }  
  
#endif  
          
    }  
    UpdateKeyStat(data);  
}  
  
char kbd\_read\_char()  
{  
    if( isEmpty())  
        return -1;  
    //현재프로세스가 다른 프로세스면 무시  
    if(cur\_foreground\_process->console != cur\_process->console)  
        return 0;  
    char ret;  
    //ret = kbd\_buf[buf\_head];  
    ret = cur\_foreground\_process->kbd\_buffer->buf[cur\_foreground\_process->kbd\_buffer->head];  
    //buf\_head = (buf\_head + 1)%BUFSIZ;  
    cur\_foreground\_process->kbd\_buffer->head = (cur\_foreground\_process->kbd\_buffer->head + 1)%BUFSIZ;  
    return ret;  
}  
  
//입력 버퍼 할당  
struct Kbd\_buffer \*get\_kbd\_buffer(){  
    int i;  
  
    for(i = 0; i < MAX\_KBD\_BUFFER; i++){  
        if(kbd\_buffer[i].used == false){  
            kbd\_buffer[i].used = true;  
            return &kbd\_buffer[i];  
        }  
    }  
  
    return NULL;  
}

1. console.c

#include <interrupt.h>  
#include <device/console.h>  
#include <type.h>  
#include <device/kbd.h>  
#include <device/io.h>  
#include <device/pit.h>  
#include <stdarg.h>  
#include <proc/proc.h>  
  
char next\_line[2]; //"\r\n";  
/\*  
#ifdef SCREEN\_SCROLL  
  
char buf\_s[SIZE\_SCROLL];   
char \*buf\_w;      
char \*buf\_p;      
  
int sum\_y;  
  
bool a\_s = TRUE;  
#endif  
\*/  
//위 전역변수를 사용하는 코드를 아래 변수를 사용하는 코드로 변경  
struct Console console[MAX\_CONSOLE\_NUM];  
extern struct process \*cur\_process;  
extern struct process \*cur\_foreground\_process;  
struct Console \*cur\_console;  
  
void init\_console(void)  
{  
    int i = 0;  
    next\_line[0] = '\r';  
    next\_line[1] = '\r';  
    //콘솔 초기화  
    for(i=0; i<MAX\_CONSOLE\_NUM; i++)  
    {  
        console[i].Glob\_x = 0;  
        console[i].Glob\_y = 0;  
        console[i].buf\_w = console[i].buf\_s;  
        console[i].buf\_p = console[i].buf\_s;  
        console[i].a\_s = TRUE;  
        console[i].sum\_y = 0;  
    }  
    //처음사용할 콘솔설정  
    cur\_process->console = &console[0];  
    cur\_console = &console[0];  
    cur\_console->Glob\_x = 0;  
    cur\_console->Glob\_y = 2;  
  
    next\_line[0] = '\r';  
    next\_line[1] = '\r';  
  
#ifdef SCREEN\_SCROLL  
    cur\_console->buf\_w = cur\_console->buf\_s;  
    cur\_console->buf\_p = cur\_console->buf\_s;  
    cur\_console->a\_s = TRUE;  
  
    cur\_console->sum\_y = 0;  
#endif  
  
}  
  
void set\_cursor(void)  
{  
    outb(0x3D4, 0x0F);  
    outb(0x3D5, (cur\_console->Glob\_y\*HSCREEN+cur\_console->Glob\_x)&0xFF);  
    outb(0x3D4, 0x0E);  
    outb(0x3D5, (((cur\_console->Glob\_y\*HSCREEN+cur\_console->Glob\_x)>>8)&0xFF));  
}  
  
void PrintCharToScreen(int x, int y, const char \*pString)   
{  
    cur\_console->Glob\_x = x;  
    cur\_console->Glob\_y = y;  
    int i = 0;  
    while(pString[i] != 0)  
    {  
        PrintChar(cur\_console->Glob\_x++, cur\_console->Glob\_y, pString[i++]);  
    }  
    cur\_console->a\_s = TRUE;  
}  
  
void PrintChar(int x, int y, const char String)   
{  
#ifdef SCREEN\_SCROLL  
    if (String == '\n') {  
        if((y+1) > VSCREEN) {  
            scroll();  
            y--;  
        }  
        cur\_console->Glob\_x = 0;  
        cur\_console->Glob\_y = y+1;  
        cur\_console->sum\_y++;  
        return;  
    }  
    else if (String == '\b') {  
        if(cur\_console->Glob\_x == 0) return;  
        cur\_console->Glob\_x-=2;  
        cur\_console->buf\_w[y \* HSCREEN + x - 1] = 0;  
    }  
    else {  
        if ((y >= VSCREEN) && (x >= 0)) {  
            scroll();  
            x = 0;  
            y--;  
        }                              
  
        char\* b = &cur\_console->buf\_w[y \* HSCREEN + x];  
        if(b >= SCROLL\_END)  
            b-= SIZE\_SCROLL;  
        \*b = String;  
  
        if(cur\_console->Glob\_x >= HSCREEN)  
        {  
            cur\_console->Glob\_x = 0;  
            cur\_console->Glob\_y++;  
            cur\_console->sum\_y++;  
        }      
    }  
#else  
    CHAR \*pScreen = (CHAR \*)VIDIO\_MEMORY;  
  
    if (String == '\n') {  
        if((y+1) > 24) {  
            scroll();  
            y--;  
        }  
        pScreen += ((y+1) \* 80);  
        cur\_console->Glob\_x = 0;  
        cur\_console->Glob\_y = y+1;  
    return;  
    }  
    else {  
        if ((y > 24) && (x >= 0)) {  
            scroll();  
            x = 0; y--;  
        }                         
  
        pScreen += ( y \* 80) + x;  
        pScreen[0].bAtt = 0x07;  
        pScreen[0].bCh = String;  
  
        if(cur\_console->Glob\_x > 79)  
        {  
            cur\_console->Glob\_x = 0;  
            cur\_console->Glob\_y++;  
        }      
    }  
#endif  
}  
  
void clrScreen(void)   
{  
    CHAR \*pScreen = (CHAR \*) VIDIO\_MEMORY;  
    int i;  
  
    for (i = 0; i < 80\*25; i++) {  
        (\*pScreen).bAtt = 0x07;  
        (\*pScreen++).bCh = ' ';  
    }     
    cur\_console->Glob\_x = 0;  
    cur\_console->Glob\_y = 0;  
}  
  
//Ctrl+l 화면 클리어 구현  
void clearScreen(void)  
{  
    char \* tmp\_buf\_p;  
    char \* tmp\_buf\_w;  
  
    cur\_console->a\_s = FALSE;  
    while(1)  
    {  
          
        tmp\_buf\_p = (char\*)((int)cur\_console->buf\_p + (HSCREEN \* 1));  
        tmp\_buf\_w = cur\_console->buf\_w + SIZE\_SCREEN;  
        if(tmp\_buf\_w > SCROLL\_END)  
            tmp\_buf\_w = (char \*)((int)tmp\_buf\_w - SIZE\_SCROLL);  
        //glob\_y가 0이 될때까지 scroll 호출하여 콘솔클리어 구현  
        if(tmp\_buf\_p > cur\_console->buf\_w && cur\_console->buf\_p <= cur\_console->buf\_w)  
            if(cur\_console->Glob\_y == 0)  
                break;  
            else  
                scroll();  
        cur\_console->buf\_p = tmp\_buf\_p;  
    }      
        if(cur\_console->buf\_p >= SCROLL\_END)  
            cur\_console->buf\_p = (char\*)((int)cur\_console->buf\_p - SIZE\_SCROLL);  
        else if(cur\_console->buf\_p < cur\_console->buf\_s)  
            cur\_console->buf\_p = (char\*)((int)cur\_console->buf\_p + SIZE\_SCROLL);  
  
    refreshScreen();  
    set\_cursor();  
}  
  
void scroll(void)   
{  
#ifdef SCREEN\_SCROLL  
    cur\_console->buf\_w += HSCREEN;  
    cur\_console->buf\_p += HSCREEN;  
      
    while(cur\_console->buf\_w > SCROLL\_END)  
        cur\_console->buf\_w -= SIZE\_SCROLL;  
  
      
    //clear line  
    int i;  
    char \*buf\_ptr = cur\_console->buf\_w + SIZE\_SCREEN;  
    for(i = 0; i < HSCREEN; i++)  
    {  
        if(buf\_ptr > SCROLL\_END)  
            buf\_ptr -= SIZE\_SCROLL;  
        \*(buf\_ptr++) = 0;  
    }  
          
//  
#else  
    CHAR \*pScreen = (CHAR \*) VIDIO\_MEMORY;  
    CHAR \*pScrBuf = (CHAR \*) (VIDIO\_MEMORY + 2\*80);  
    int i;  
    for (i = 0; i < 80\*24; i++) {  
            (\*pScreen).bAtt = (\*pScrBuf).bAtt;  
                (\*pScreen++).bCh = (\*pScrBuf++).bCh;  
    }     
    for (i = 0; i < 80; i++) {  
            (\*pScreen).bAtt = 0x07;  
                (\*pScreen++).bCh = ' ';  
    }   
#endif  
    cur\_console->Glob\_y--;  
  
}  
  
#ifdef SERIAL\_STDOUT  
void printCharToSerial(const char \*pString)  
{  
    int i;  
    enum intr\_level old\_level = intr\_disable();  
    for(;\*pString != NULL; pString++)  
    {  
        if(\*pString != '\n'){  
            while((inb(LINE\_STATUS) & THR\_EMPTY) == 0)  
                continue;  
            outb(FIRST\_SPORT, \*pString);  
  
        }  
        else{  
            for(i=0; i<2; i++){  
                while((inb(LINE\_STATUS) & THR\_EMPTY) == 0)  
                    continue;  
                outb(FIRST\_SPORT, next\_line[i]);  
            }  
        }  
    }  
    intr\_set\_level (old\_level);  
}  
#endif  
  
  
int printk(const char \*fmt, ...)  
{  
    char buf[1024];  
    va\_list args;  
    int len;  
  
    va\_start(args, fmt);  
    len = vsprintk(buf, fmt, args);  
    va\_end(args);  
      
#ifdef SERIAL\_STDOUT  
    printCharToSerial(buf);  
#endif  
    //호출한 프로세스의 콘솔이 현재 사용하는 콘솔일 때까지 기다림   
    while(1)  
    {  
          
        if(cur\_process->console == cur\_console)  
        {  
            PrintCharToScreen(cur\_console->Glob\_x, cur\_console->Glob\_y, buf);  
            break;  
        }  
    }  
    return len;  
}  
  
#ifdef SCREEN\_SCROLL  
void scroll\_screen(int offset)  
{  
    char \* tmp\_buf\_p;  
    char \* tmp\_buf\_w;  
    if(cur\_console->a\_s == TRUE && offset > 0 && cur\_console->buf\_p == cur\_console->buf\_w)  
        return;  
  
    cur\_console->a\_s = FALSE;  
      
    tmp\_buf\_p = (char\*)((int)cur\_console->buf\_p + (HSCREEN \* offset));  
    tmp\_buf\_w = cur\_console->buf\_w + SIZE\_SCREEN;  
    if(tmp\_buf\_w > SCROLL\_END)  
        tmp\_buf\_w = (char \*)((int)tmp\_buf\_w - SIZE\_SCROLL);  
  
    if(cur\_console->sum\_y < NSCROLL && offset < 0 && tmp\_buf\_p <= cur\_console->buf\_s && cur\_console->buf\_p > cur\_console->buf\_s) return;// 처음  
    if(offset > 0 && tmp\_buf\_p > cur\_console->buf\_w && cur\_console->buf\_p <= cur\_console->buf\_w) return;  
    else if(offset < 0 && tmp\_buf\_p <= tmp\_buf\_w && cur\_console->buf\_p > tmp\_buf\_w) return;  
  
    cur\_console->buf\_p = tmp\_buf\_p;  
      
    if(cur\_console->buf\_p >= SCROLL\_END)  
        cur\_console->buf\_p = (char\*)((int)cur\_console->buf\_p - SIZE\_SCROLL);  
    else if(cur\_console->buf\_p < cur\_console->buf\_s)  
        cur\_console->buf\_p = (char\*)((int)cur\_console->buf\_p + SIZE\_SCROLL);  
  
    refreshScreen();  
}  
  
void set\_fallow(void)  
{  
    cur\_console->a\_s = TRUE;  
}  
  
void refreshScreen(void)  
{  
    CHAR \*p\_s= (CHAR \*) VIDIO\_MEMORY;  
    int i;  
  
    if(cur\_console->a\_s)  
        cur\_console->buf\_p = cur\_console->buf\_w;  
  
    char\* b = cur\_console->buf\_p;  
  
    for(i = 0; i < SIZE\_SCREEN; i++, b++, p\_s++)  
    {  
        if(b >= SCROLL\_END)  
            b -= SIZE\_SCROLL;  
        p\_s->bAtt = 0x07;  
        p\_s->bCh = \*b;  
    }  
}  
  
//콘솔 할당 함수  
struct console \*get\_console(void){  
    int i;  
  
    for(i = 0; i < MAX\_CONSOLE\_NUM; i++){  
        if(console[i].used == FALSE){  
            console[i].used = TRUE;  
            return &console[i];  
        }  
    }  
  
    return NULL;  
}  
#endif

1. proc.c

#include <list.h>  
#include <proc/sched.h>  
#include <mem/malloc.h>  
#include <proc/proc.h>  
#include <ssulib.h>  
#include <interrupt.h>  
#include <proc/sched.h>  
#include <device/console.h>  
#include <device/io.h>  
#include <syscall.h>  
#include <mem/paging.h>  
#include <mem/palloc.h>  
#include <string.h>  
  
#define STACK\_SIZE 512  
#define PROC\_NUM\_MAX 16  
  
struct list p\_list;        // All Porcess List  
struct list r\_list;        // Run Porcess List  
struct list s\_list;        // Sleep Process List  
struct list d\_list;        // Deleted Process List   
struct list f\_list;        // Foreground Process List  
  
struct process procs[PROC\_NUM\_MAX];  
struct process \*cur\_process;  
struct process \*cur\_foreground\_process;  
extern struct Console \*cur\_console;  
int pid\_num\_max;  
  
uint32\_t process\_stack\_ofs;  
  
//values for pid  
static int lock\_pid\_simple; //1 : lock, 0 : unlock  
static int lately\_pid;        //init vaule = -1  
  
bool more\_prio(const struct list\_elem \*a, const struct list\_elem \*b,void \*aux);  
bool less\_time\_sleep(const struct list\_elem \*a, const struct list\_elem \*b,void \*aux);  
pid\_t getValidPid(int \*idx);  
  
void proc\_start(void);              
void proc\_end(void);              
  
static void login\_prompt(void \*);  
static bool check\_user(char \*, char \*);  
void shell\_proc(void\* aux);  
  
typedef struct{  
    char id[BUFSIZ];  
    char password[BUFSIZ];  
}user\_list;  
  
user\_list temp\_list = {"ssuos\n","oslab\n"};  
  
void loop\_proc(void \*aux)  
{  
    printk("loop start...\n");  
    while(1);  
}  
  
void login\_prompt(void \* aux)  
{  
    char id[BUFSIZ];  
    char password[BUFSIZ];  
  
    while(1)  
    {  
        printk("\nid : ");  
        while(getkbd(id,BUFSIZ) == TRUE);  
  
        printk("password : ");  
        while(getkbd(password,BUFSIZ) == TRUE);  
  
        if(check\_user(id,password))  
            shell\_proc(NULL);  
        else  
            printk("\nincorrect id or password.\n");  
    }  
}  
  
bool check\_user(char \*id, char \*password)  
{  
    if (strcmp([temp\_list.id](http://temp_list.id/),id) || strcmp(temp\_list.password,password)) return false;  
    return true;  
}  
  
  
void init\_proc()  
{  
    process\_stack\_ofs = offsetof (struct process, stack);  
  
    lock\_pid\_simple = 0;  
    lately\_pid = -1;  
  
    list\_init(&p\_list);  
    list\_init(&r\_list);  
    list\_init(&s\_list);  
    list\_init(&d\_list);  
    list\_init(&f\_list);  
  
    int i;  
    for (i = 0; i < PROC\_NUM\_MAX; i++)  
    {  
        procs[i].pid = i;  
        procs[i].state = PROC\_UNUSED;  
        procs[i].parent = NULL;  
    }  
  
    pid\_t pid = getValidPid(&i);  
    cur\_process = &procs[i];  
  
    cur\_process->pid = pid;  
    cur\_process->parent = NULL;  
    cur\_process->state = PROC\_RUN;  
    cur\_process->priority = 0;  
    cur\_process->stack = 0;  
    cur\_process->pd = (void\*)read\_cr3();  
    cur\_process->elem\_all.prev = NULL;  
    cur\_process->elem\_all.next = NULL;  
    cur\_process->elem\_stat.prev = NULL;  
    cur\_process->elem\_stat.next = NULL;  
    //f\_list에 넣기 위해 초기화  
    cur\_process->elem\_foreground.prev = NULL;  
    cur\_process->elem\_foreground.next = NULL;  
    //콘솔, 키보드버퍼 등록  
    cur\_process->kbd\_buffer = get\_kbd\_buffer();  
    cur\_process->console = get\_console();  
    //foreground프로세스와 콘솔을 현재 프로세스로 바꿔줌  
    cur\_foreground\_process = cur\_process;  
    cur\_console = cur\_process->console;  
      
    list\_push\_back(&f\_list, &cur\_process->elem\_foreground);//f\_list에 넣음  
    list\_push\_back(&p\_list, &cur\_process->elem\_all);  
    list\_push\_back(&r\_list, &cur\_process->elem\_stat);  
}  
  
pid\_t getValidPid(int \*idx) {  
  
    pid\_t pid = -1;  
    int i;  
  
    while(lock\_pid\_simple)  
        ;  
  
    lock\_pid\_simple++;  
  
    // find unuse process pid and return it  
    for(i = 0; i < PROC\_NUM\_MAX; i++)  
    {  
        int tmp = i + lately\_pid + 1;// % PROC\_NUM\_MAX;  
        if(procs[tmp % PROC\_NUM\_MAX].state == PROC\_UNUSED) { // find out valid state;  
            pid = lately\_pid + 1;  
            \*idx = tmp % PROC\_NUM\_MAX;  
            break;  
        }  
    }  
  
    if(pid != -1)  
    {  
        lately\_pid = pid;      
    }  
  
    lock\_pid\_simple = 0;  
  
    return pid;  
}  
  
pid\_t proc\_create(proc\_func func, struct proc\_option \*opt, void\* aux)  
{  
    struct process \*p;  
    int idx;  
  
    enum intr\_level old\_level = intr\_disable();  
  
    pid\_t pid = getValidPid(&idx);  
    p = &procs[idx];  
  
    p->pid = pid;  
    p->state = PROC\_RUN;  
  
    if(opt != NULL)  
        p->priority = opt->priority;     
    else  
        p->priority = (unsigned char)0;  
  
    p->time\_used = 0;  
    p->time\_sched= 0;  
    p->parent = cur\_process;  
    p->simple\_lock = 0;  
    p->child\_pid = -1;  
    p->pd = pd\_create(pid);  
  
    //init stack  
    int \*top = (int\*)palloc\_get\_page();  
    int stack = (int)top;  
    top = (int\*)stack + STACK\_SIZE - 1;  
  
    \*(--top) = (int)aux;        //argument for func  
    \*(--top) = (int)proc\_end;    //return address from func  
    \*(--top) = (int)func;        //return address from proc\_start  
    \*(--top) = (int)proc\_start; //return address from switch\_process  
  
  
    \*(--top) = (int)((int\*)stack + STACK\_SIZE - 1); //ebp  
    \*(--top) = 1; //eax  
    \*(--top) = 2; //ebx  
    \*(--top) = 3; //ecx  
    \*(--top) = 4; //edx  
    \*(--top) = 5; //esi  
    \*(--top) = 6; //edi  
  
    p->stack = top;  
    p->elem\_all.prev = NULL;  
    p->elem\_all.next = NULL;  
    p->elem\_stat.prev = NULL;  
    p->elem\_stat.next = NULL;  
    p->elem\_foreground.prev = NULL;  
    p->elem\_foreground.next = NULL;  
  
  
    //check option, set Console & Kbd  
    //list element, kbd\_buffer, console  
    if(opt!=NULL&&opt->foreground)//foreground 옵션일 때  
    {  
        p->console = get\_console();  
        cur\_console = p->console;  
        p->kbd\_buffer = get\_kbd\_buffer();  
        list\_push\_back(&f\_list, &p->elem\_foreground);  
        cur\_foreground\_process = p;  
    }  
    else//background 옵션일 때  
    {  
        p->console = p->parent->console;  
        cur\_console = p->console;  
        p->kbd\_buffer = NULL;  
    }  
      
  
    list\_push\_back(&p\_list, &p->elem\_all);  
    list\_push\_back(&r\_list, &p->elem\_stat);  
  
    intr\_set\_level (old\_level);  
    return p->pid;  
}  
  
void\* getEIP()  
{  
    return \_\_builtin\_return\_address(0);  
}  
  
void  proc\_start(void)  
{  
    intr\_enable ();  
    return;  
}  
  
void proc\_free(void)  
{  
    uint32\_t pt = \*(uint32\_t\*)cur\_process->pd;  
    cur\_process->parent->child\_pid = cur\_process->pid;  
    cur\_process->parent->simple\_lock = 0;  
  
    list\_remove(&cur\_process->elem\_stat);  
  
    cur\_process->state = PROC\_ZOMBIE;    //change state  
    list\_push\_back(&d\_list, &cur\_process->elem\_stat);  
  
    // free page  
    palloc\_free\_page(cur\_process->stack);  
    palloc\_free\_page((void\*)pt);  
    palloc\_free\_page(cur\_process->pd);  
}  
  
void proc\_end(void)  
{  
    proc\_free();  
    schedule();  
    printk("never reach\n");  
    return;    //never reach  
}  
  
void proc\_wake(void)  
{  
    struct process\* p;  
    unsigned long long t = get\_ticks();  
  
    while(!list\_empty(&s\_list))  
    {  
        p = list\_entry(list\_front(&s\_list), struct process, elem\_stat);  
        if(p->time\_sleep > t)  
            break;  
        //proc\_unblock(p);  
        p->state = PROC\_RUN;  
        list\_remove(&p->elem\_stat);  
    }  
}  
  
void proc\_sleep(unsigned ticks)  
{  
    unsigned long cur\_ticks = get\_ticks();  
    cur\_process->time\_sleep =  ticks + cur\_ticks;  
    cur\_process->state = PROC\_STOP;  
    list\_insert\_ordered(&s\_list, &cur\_process->elem\_stat,  
            less\_time\_sleep, NULL);  
    schedule();  
}  
  
void proc\_block(void)  
{  
    cur\_process->state = PROC\_BLOCK;  
    schedule();      
}  
  
void proc\_unblock(struct process\* proc)  
{  
    enum intr\_level old\_level;  
  
    old\_level = intr\_disable();  
  
    list\_push\_back(&r\_list, &proc->elem\_stat);  
    proc->state = PROC\_RUN;  
  
    intr\_set\_level(old\_level);  
}       
  
bool less\_time\_sleep(const struct list\_elem \*a, const struct list\_elem \*b,void \*aux)  
{  
    struct process \*p1 = list\_entry(a, struct process, elem\_stat);  
    struct process \*p2 = list\_entry(b, struct process, elem\_stat);  
  
    return p1->time\_sleep < p2->time\_sleep;  
}  
  
bool more\_prio(const struct list\_elem \*a, const struct list\_elem \*b,void \*aux)  
{  
    struct process \*p1 = list\_entry(a, struct process, elem\_stat);  
    struct process \*p2 = list\_entry(b, struct process, elem\_stat);  
    return p1->priority > p2->priority;  
}  
  
  
void kernel1\_proc(void\* aux)  
{  
    cur\_process -> priority = 200;  
    while(1)  
    {  
        schedule();  
    }  
}  
  
void kernel2\_proc(void\* aux)  
{  
    cur\_process -> priority = 200;  
    while(1)  
    {  
        schedule();  
    }  
}  
  
void ps\_proc(void\* aux)  
{  
    int i;  
    for(i = 0; i<PROC\_NUM\_MAX; i++)  
    {  
        struct process \*p = &procs[i];  
  
        if(p->state == PROC\_UNUSED)  
            continue;  
  
        printk("pid %d ppid ", p->pid);  
  
        if(p->parent != NULL)  
            printk("%d", p->parent->pid);  
        else  
            printk("non");  
  
        printk(" state %d prio %d using time %d sched time %d\n",  
                p->state, p->priority, p->time\_used, p->time\_sched);  
  
    }  
    exit(1);  
}  
  
extern const char\* VERSION;  
extern const char\* AUTHOR;  
extern const char\* MODIFIER;  
void uname\_proc(void\* aux)  
{  
    printk("SSUOS %s\nmade by %s\nmodefied by %s\n", VERSION, AUTHOR, MODIFIER);      
  
}  
  
void create\_shell\_proc(void\* aux)  
{  
    struct proc\_option proc\_opt = {0, TRUE};  
    proc\_create(login\_prompt,&proc\_opt,NULL);  
}  
  
void print\_pid(void\* aux) {  
  
    while(1) {  
        printk("pid = %d ", cur\_process->pid);  
        printk("prio = %d ", cur\_process->priority);  
        printk("time = %d ", cur\_process->time\_slice);  
        printk("ticks = %d ", get\_ticks());  
        printk("in %s\n", aux);  
  
#define SLEEP\_FREQ 100  
        proc\_sleep(cur\_process->pid \* cur\_process->pid \* SLEEP\_FREQ);  
    }  
}  
  
typedef struct  
{  
    char\* cmd;  
    unsigned char type;  
    void\* func;  
} CmdList;  
  
void shell\_proc(void\* aux)  
{  
    CmdList cmdlist[] = {  
        {"shutdown", 0, shutdown},  
        {"ps", 1, ps\_proc},  
        {"uname", 1, uname\_proc},  
        {"create\_shell", 0, create\_shell\_proc},  
    };  
#define CMDNUM 4  
#define TOKNUM 10  
    char buf[BUFSIZ];  
    char token[TOKNUM][BUFSIZ];  
    int token\_num;  
  
    cur\_process -> priority = 100;  
  
    while(1)  
    {  
        proc\_func \*func;  
        int i, len;  
  
        printk("> ");  
  
        while(getkbd(buf,BUFSIZ))  
        {  
            ;   
        }  
          
        for(i=0;buf[i] != '\n'; i++);   
        for(i--; buf[i] == ' '; i--)  
            buf[i] = 0;  
  
        token\_num = getToken(buf,token,TOKNUM);  
  
        if( strcmp(token[0], "exit") == 0)  
            break;  
  
        if( strncmp(token[0], "list", BUFSIZ) == 0)  
        {  
            for(i = 0; i < CMDNUM; i++)  
                printk("%s\n", cmdlist[i].cmd);  
            continue;  
        }  
          
        for(i = 0; i < CMDNUM; i++)  
        {  
            if( strncmp(cmdlist[i].cmd, token[0], BUFSIZ) == 0)  
                break;  
        }  
  
        if(i == CMDNUM)  
        {  
            printk("Unknown command %s\n", buf);  
            continue;  
        }  
  
        if(cmdlist[i].type == 0)  
        {  
            void (\*func)(void);  
            func = cmdlist[i].func;  
            func();  
        }  
        else if(cmdlist[i].type == 1)  
        {  
            cur\_process->simple\_lock = 1;  
            int pid = fork(cmdlist[i].func, (void\*)0x999);  
  
            while(cur\_process->simple\_lock)  
                ;  
        }  
        else  
        {  
            printk("Unknown type\n");  
            continue;  
        }  
    }  
}  
  
void idle(void\* aux)  
{  
      
    proc\_create(kernel1\_proc, NULL, NULL);  
    proc\_create(kernel2\_proc, NULL, NULL);  
    proc\_create(login\_prompt,NULL,NULL);  
  
    while(1) {    
        if(cur\_process->pid != 0) {  
            printk("error : idle process's pid != 0\n", cur\_process->pid);  
            while(1);  
        }  
  
        while( !list\_empty(&d\_list) )  
        {  
            struct list\_elem \*e = list\_pop\_front(&d\_list);  
            struct process \*p = list\_entry(e, struct process, elem\_stat);  
            p->state = PROC\_UNUSED;  
            list\_remove( &p->elem\_all);  
        }  
  
        schedule();       
    }  
}  
  
void proc\_print\_data()  
{  
    int a, b, c, d, bp, si, di, sp;  
  
    //eax ebx ecx edx  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(a));  
  
    \_\_asm\_\_ \_\_volatile("mov %ebx ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(b));  
      
    \_\_asm\_\_ \_\_volatile("mov %ecx ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(c));  
      
    \_\_asm\_\_ \_\_volatile("mov %edx ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(d));  
      
    //ebp esi edi esp  
    \_\_asm\_\_ \_\_volatile("mov %ebp ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(bp));  
  
    \_\_asm\_\_ \_\_volatile("mov %esi ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(si));  
  
    \_\_asm\_\_ \_\_volatile("mov %edi ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(di));  
  
    \_\_asm\_\_ \_\_volatile("mov %esp ,%eax");  
    \_\_asm\_\_ \_\_volatile("mov %%eax ,%0": "=m"(sp));  
  
    printk(    "\neax %o ebx %o ecx %o edx %o"\  
            "\nebp %o esi %o edi %o esp %o\n"\  
            , a, b, c, d, bp, si, di, sp);  
}  
  
void next\_foreground\_proc(void){  
    struct list\_elem \*e;  
  
    //kbd  
    e = &cur\_foreground\_process->elem\_foreground;  
  
    if(e == list\_rbegin(&f\_list))  
        e = list\_begin(&f\_list);  
    else  
        e = list\_next(e);  
  
    cur\_foreground\_process = list\_entry(e, struct process, elem\_foreground);  
    //console  
    cur\_console = cur\_foreground\_process->console;  
    //그냥 넣어봄      
    refreshScreen();  
    set\_cursor();  
}  
  
void hexDump (void \*addr, int len) {  
    int i;  
    unsigned char buff[17];  
    unsigned char \*pc = (unsigned char\*)addr;  
  
    if (len == 0) {  
        printk("  ZERO LENGTH\n");  
        return;  
    }  
    if (len < 0) {  
        printk("  NEGATIVE LENGTH: %i\n",len);  
        return;  
    }  
  
    // Process every byte in the data.  
    for (i = 0; i < len; i++) {  
        // Multiple of 16 means new line (with line offset).  
  
        if ((i % 16) == 0) {  
            // Just don't print ASCII for the zeroth line.  
            if (i != 0)  
                printk ("  %s\n", buff);  
  
            // Output the offset.  
            printk ("  %04x ", i);  
        }  
  
        // Now the hex code for the specific character.  
        printk (" %02x", pc[i]);  
  
        // And store a printable ASCII character for later.  
        if ((pc[i] < 0x20) || (pc[i] > 0x7e))  
            buff[i % 16] = '.';  
        else  
            buff[i % 16] = pc[i];  
        buff[(i % 16) + 1] = '\0';  
    }  
  
    // Pad out last line if not exactly 16 characters.  
    while ((i % 16) != 0) {  
        printk ("   ");  
        i++;  
    }  
  
    // And print the final ASCII bit.  
    printk ("  %s\n", buff);  
}

1. ssulib.c

#include <ssulib.h>  
#include <device/console.h>  
#include <device/io.h>  
#include <syscall.h>  
#include <proc/proc.h>  
  
extern struct process \*cur\_foreground\_process;  
extern struct process \*cur\_process;  
  
  
int strncmp(char\* b1, char\* b2, int len)  
{  
    int i;  
  
    for(i = 0; i < len; i++)  
    {  
        char c = b1[i] - b2[i];  
        if(c)  
            return c;  
        if(b1[i] == 0)  
            return 0;  
    }  
    return 0;  
}  
  
bool getkbd(char \*buf, int len)   
{  
    char ch;  
    int offset = 0;  
    struct process \*p = cur\_process;//호출한 프로세스 저장  
      
    len--;  
  
    for(; offset < len && buf[offset] != 0; offset++)  
        if(buf[offset] == '\n')  
        {  
            for(;offset>=0;offset--)  
                buf[offset] = 0;  
            offset++;  
            break;  
        }  
          
    while (1)  
    {  
        //키보드버퍼가 다르면 무시  
        if(p->kbd\_buffer!= cur\_process->kbd\_buffer)  
            continue;  
        else  
            ch = ssuread();  
        if(ch < 0)  
            break;  
        if(ch == '\b' && offset == 0)  
        {  
            set\_cursor();  
            continue;  
        }  
        if(p->console == cur\_process->console)  
        {      
            printk("%c",ch);  
            set\_cursor();  
        }  
        if (ch == '\b')  
        {  
            buf[offset-1] = 0;  
            offset--;  
        }  
        else if (ch == '\n')  
        {  
            buf[offset] = ch;  
            return FALSE;  
        }  
        else  
        {  
            buf[offset] = ch;  
            offset++;  
        }  
        if(offset == len) offset--;  
          
    }  
  
    return TRUE;  
}  
  
int getToken(char\* buf, char token[][BUFSIZ], int max\_tok)  
{  
    int num\_tok = 0;  
    int off\_tok = 0;  
    while(num\_tok < max\_tok && \*buf != '\n')  
    {  
        if(\*buf == ' ')   
        {  
            token[num\_tok][off\_tok] = 0;  
            while(\*buf == ' ') buf++;  
            off\_tok = 0;  
            num\_tok++;  
        }  
        else  
        {  
            token[num\_tok][off\_tok] = \*buf;  
            off\_tok++;  
            buf++;  
        }  
    }  
    token[num\_tok][off\_tok] = 0;  
    num\_tok++;  
  
    return num\_tok;  
}