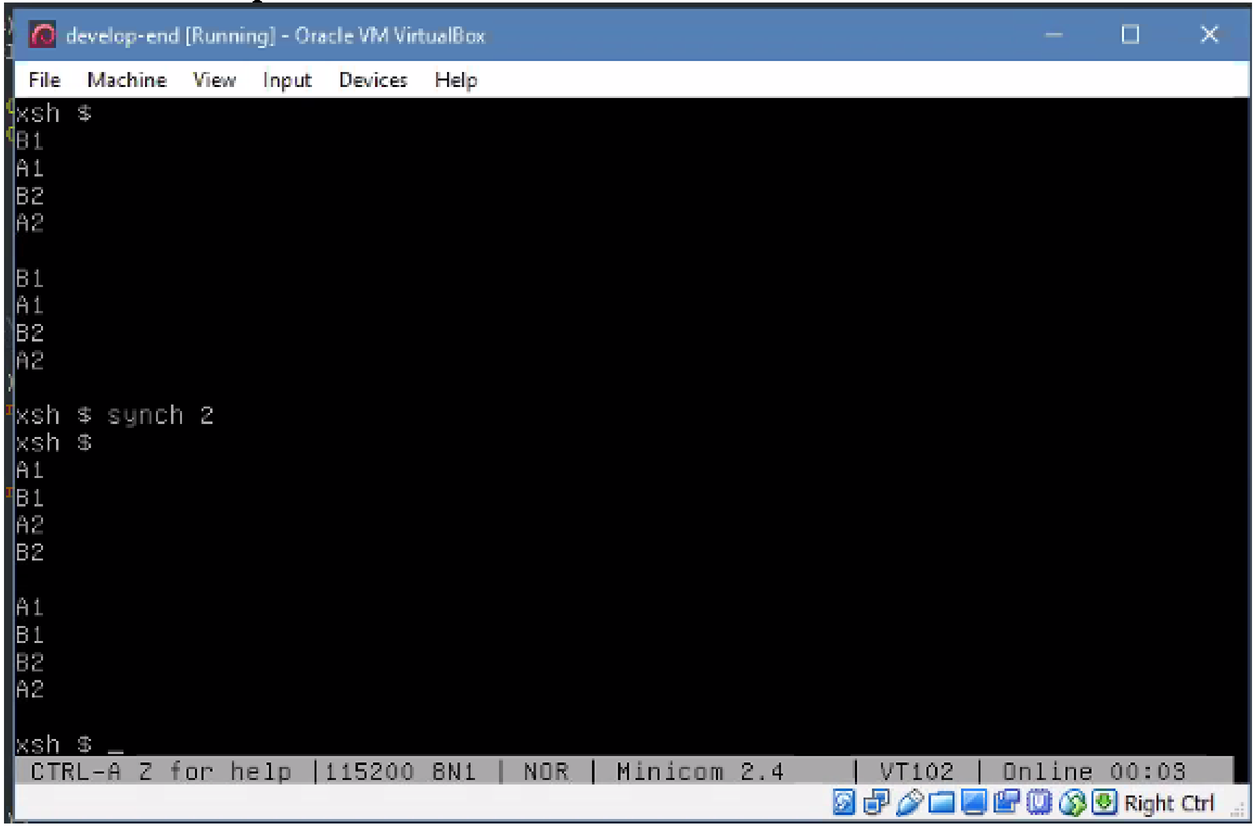
LAB4

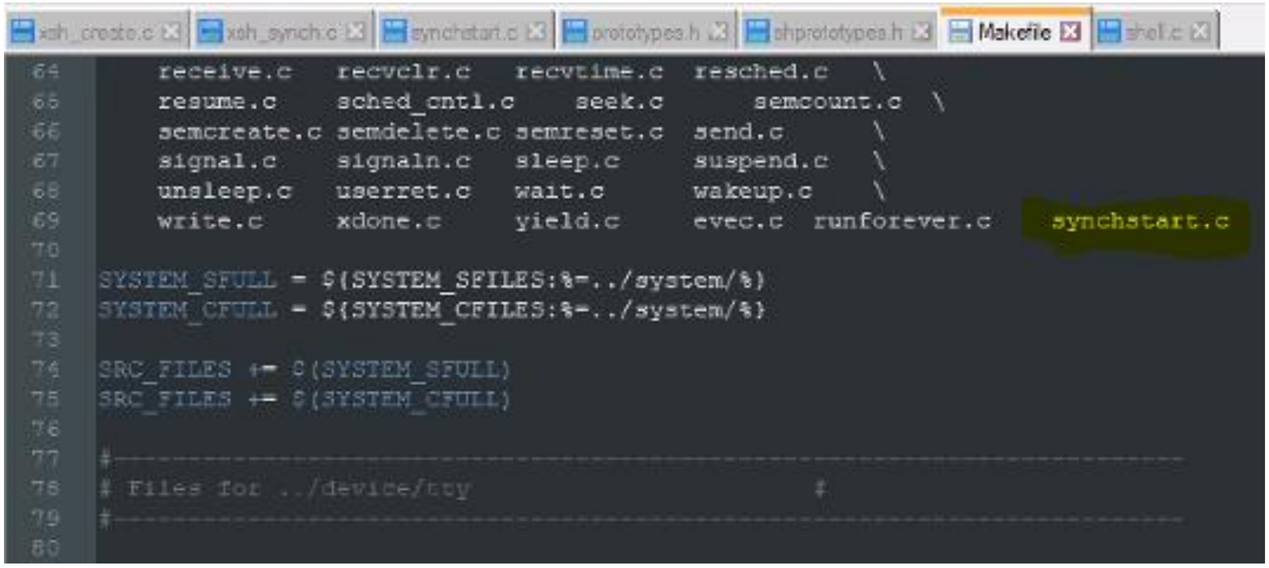
Console Output


develop-end [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
Right Ctrl

```
xsh $
B1
A1
B2
A2

B1
A1
B2
A2

xsh $ synch 2
xsh $
A1
B1
A2
B2

A1
B1
B2
A2

xsh $ _
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.4     | VT102 | Online 00:03
```

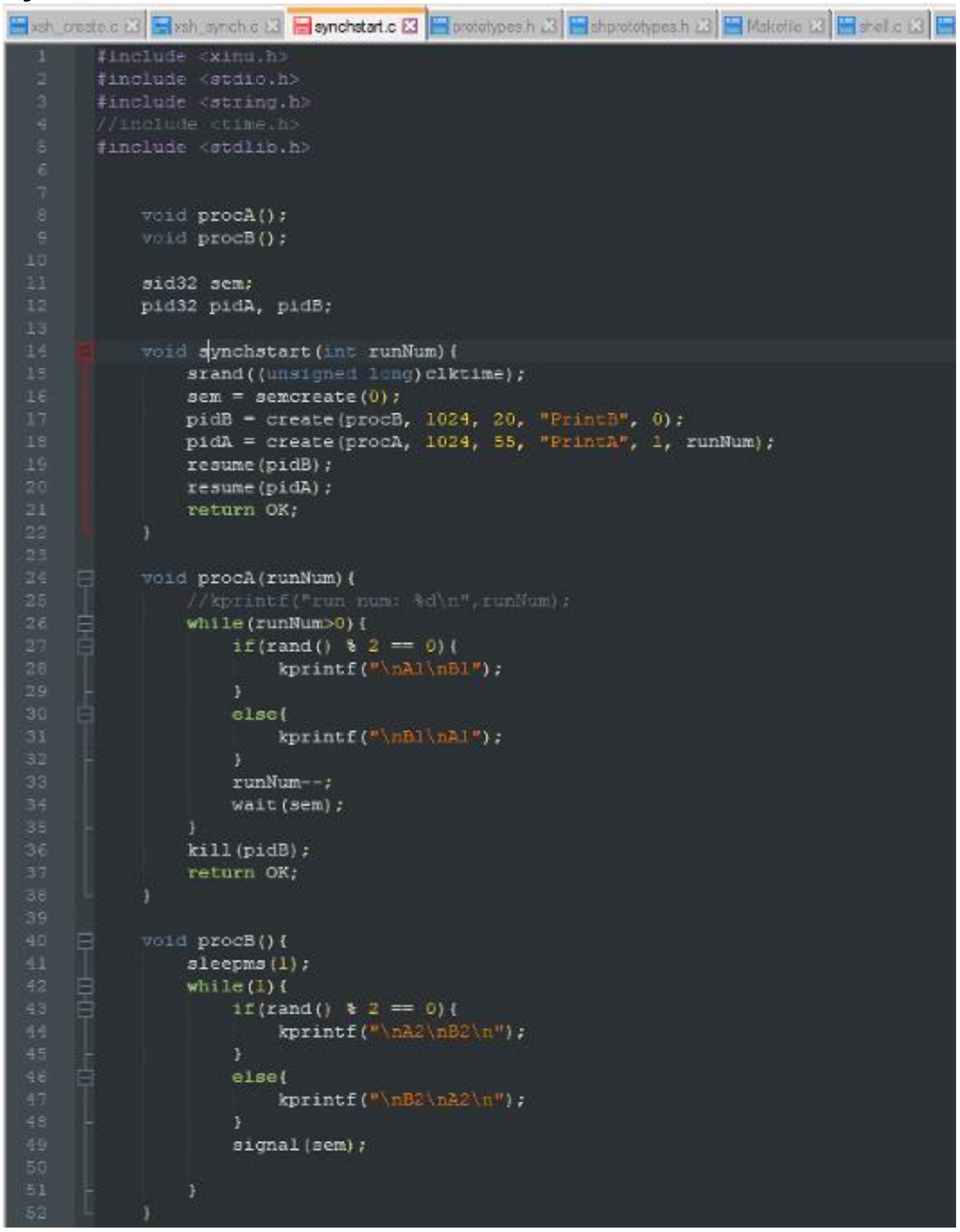
The objective for this exercise was to create two processes and coordinate them using semaphores to achieve rendezvous. Using the included random value generator, and the clock counter as a random seed, we provide a dynamic way to generate the acceptable order of execution sequences between process A, and process B. This is fully abstracted from the user, and an optional integer argument following the shell command call allows for multiple process cycles to occur without reissuing the shell command. In the above image we show that the outputs for process A and process B follow the assignment guidelines, produce the desired outcome and are not hardcoded. Screenshots from the modified files for this lab are provided.

Makefile



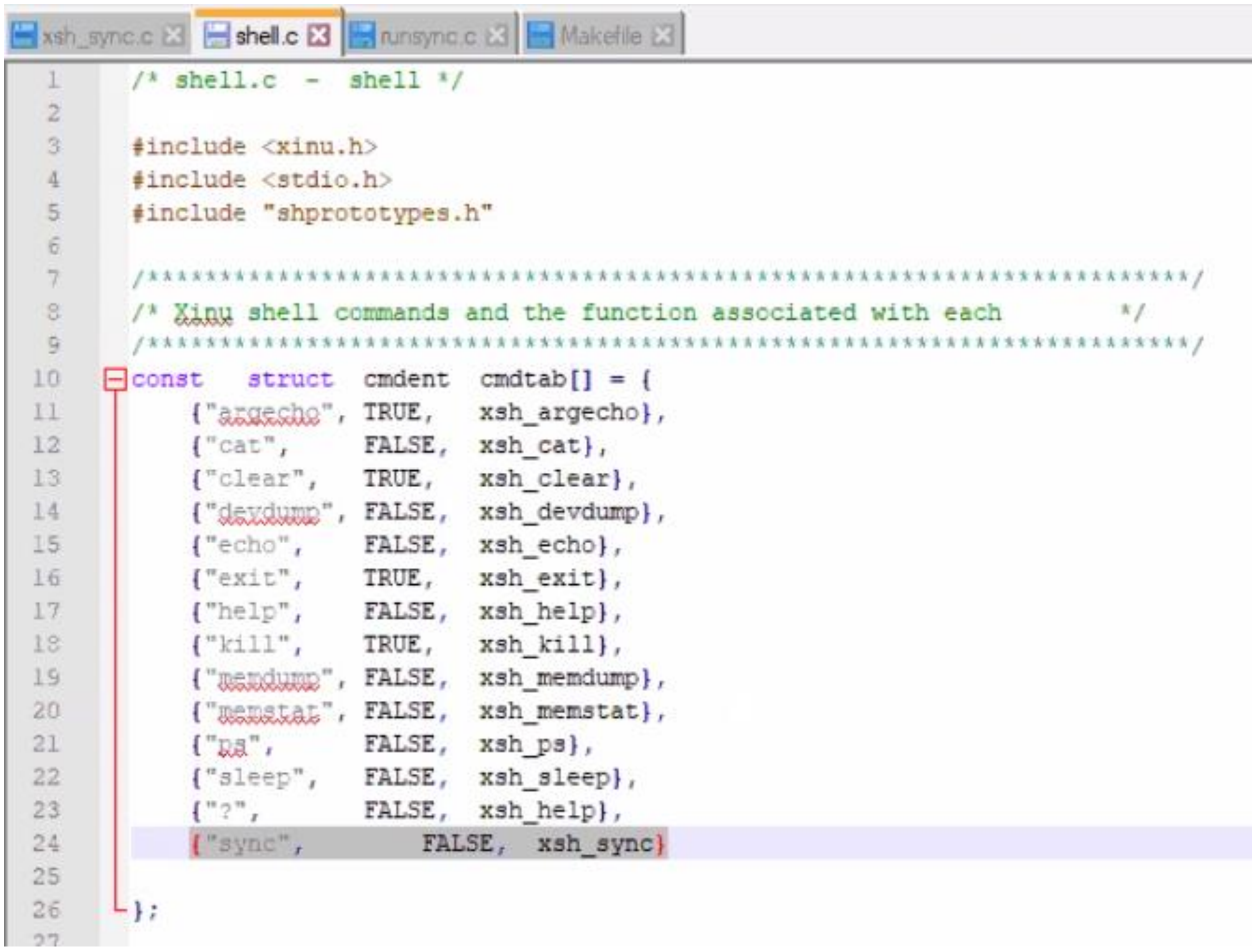
```
    receive.c   recvclr.c   recvtime.c  resched.c   \
    resume.c    sched_cntl.c    seek.c      semcount.c  \
    semcreate.c semdelete.c semreset.c  send.c      \
    signal.c    signaln.c   sleep.c     suspend.c   \
    unsleep.c   userret.c   wait.c      wakeup.c    \
    write.c     xdone.c     yield.c     evec.c  runforever.c    synchstart.c

SYSTEM_SFULL = ${SYSTEM_SFILES:%=../system/%}
SYSTEM_CFULL = ${SYSTEM_CFILES:%=../system/%}

SRC_FILES += $(SYSTEM_SFULL)
SRC_FILES += $(SYSTEM_CFULL)

#------------------------------------------------------------------------
# Files for ../device/tty                           #
#------------------------------------------------------------------------

```

synchstart.c



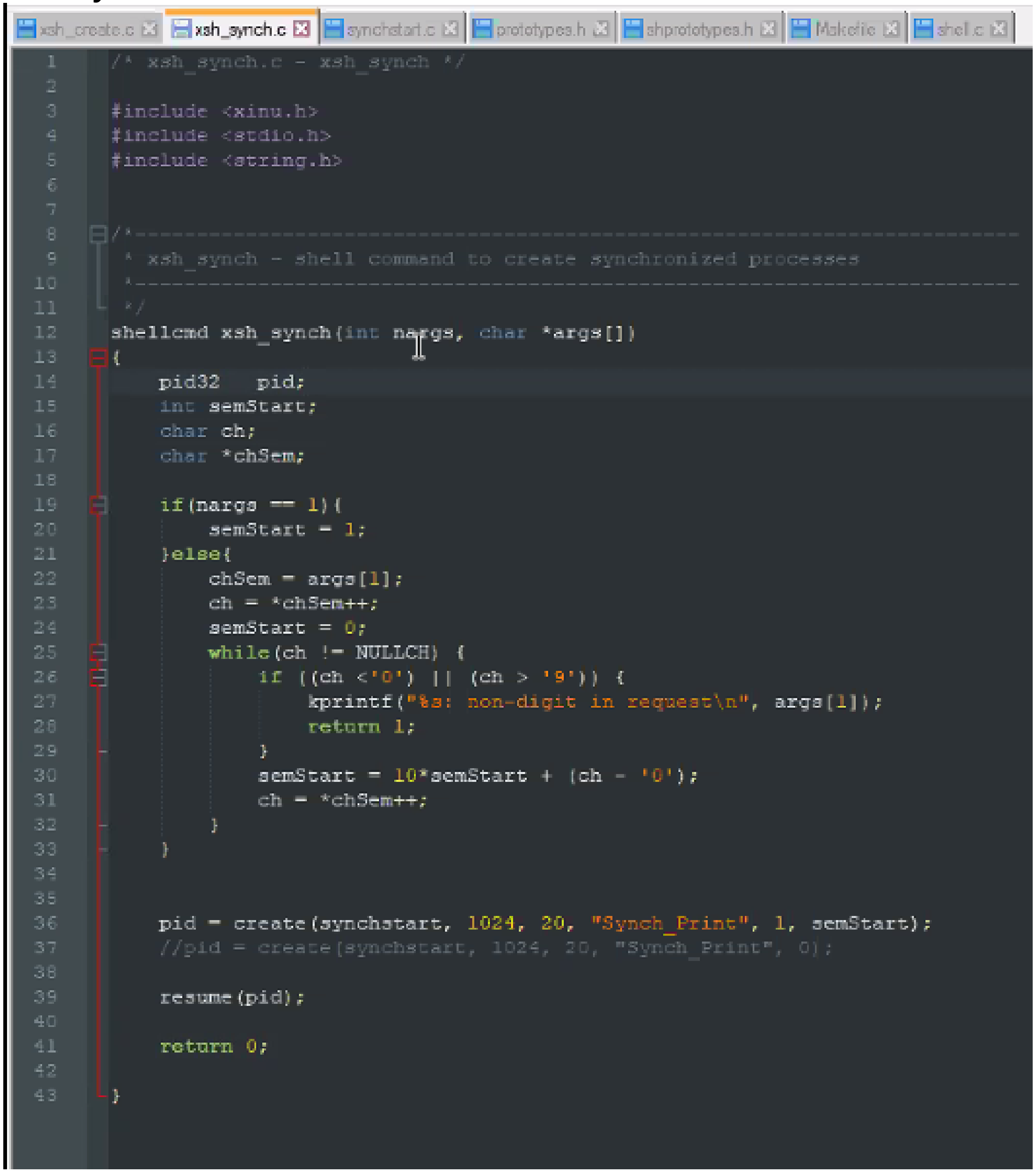
```c
#include <xinu.h>
#include <stdio.h>
#include <string.h>
//include <time.h>
#include <stdlib.h>


    void procA();
    void procB();

    sid32 sem;
    pid32 pidA, pidB;

    void synchstart(int runNum){
        srand((unsigned long)clktime);
        sem = semcreate(0);
        pidB = create(procB, 1024, 20, "PrintB", 0);
        pidA = create(procA, 1024, 55, "PrintA", 1, runNum);
        resume(pidB);
        resume(pidA);
        return OK;
    }

    void procA(runNum){
        //kprintf("run num: %d\n",runNum);
        while(runNum>0){
            if(rand() % 2 == 0){
                kprintf("\nA1\nB1");
            }
            else{
                kprintf("\nB1\nA1");
            }
            runNum--;
            wait(sem);
        }
        kill(pidB);
        return OK;
    }

    void procB(){
        sleepms(1);
        while(1){
            if(rand() % 2 == 0){
                kprintf("\nA2\nB2\n");
            }
            else{
                kprintf("\nB2\nA2\n");
            }
            signal(sem);

        }
    }
```

shell.c



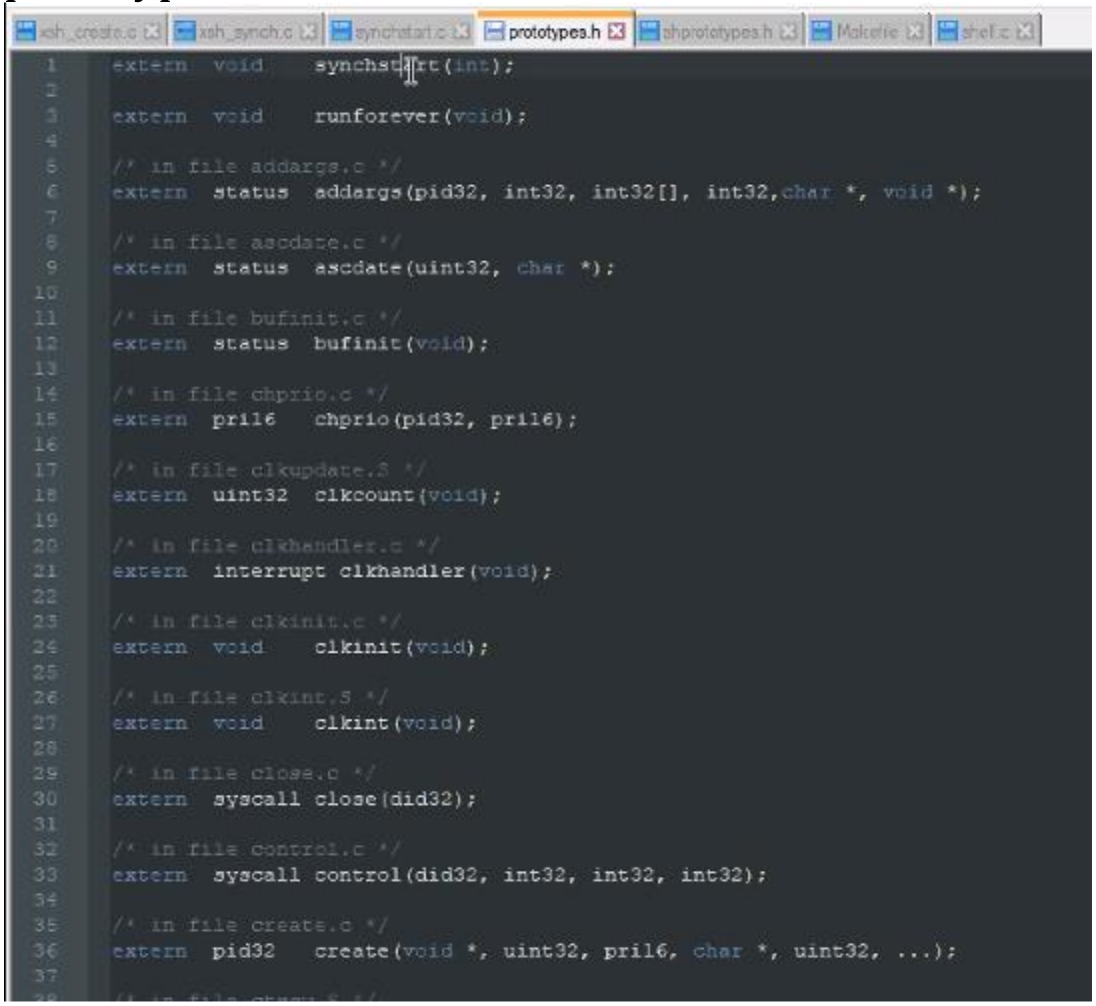
```c
/* shell.c  -  shell */

#include <xinu.h>
#include <stdio.h>
#include "shprototypes.h"

/************************************************************************/
/* Xinu shell commands and the function associated with each         */
/************************************************************************/
const   struct  cmdent  cmdtab[] = {
    {"argecho", TRUE,   xsh_argecho},
    {"cat",     FALSE,  xsh_cat},
    {"clear",   TRUE,   xsh_clear},
    {"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},
    {"ps",      FALSE,  xsh_ps},
    {"sleep",   FALSE,  xsh_sleep},
    {"?",       FALSE,  xsh_help},
    {"sync",        FALSE,  xsh_sync}

};
```

xsh_sync.c



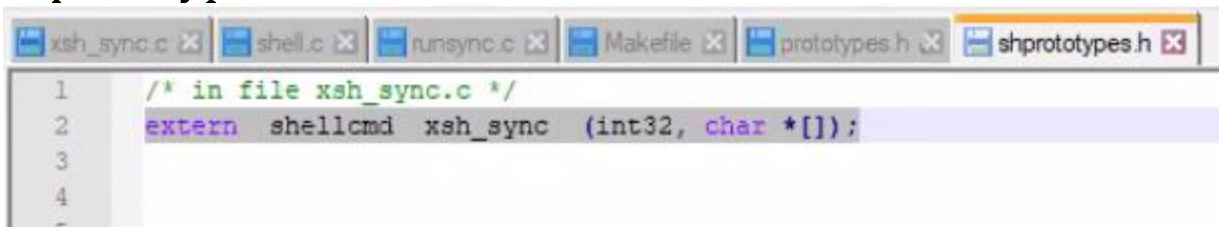
```c
/* xsh_synch.c - xsh_synch */

#include <xinu.h>
#include <stdio.h>
#include <string.h>


/*------------------------------------------------------------------------
 * xsh_synch - shell command to create synchronized processes
 *------------------------------------------------------------------------
 */
shellcmd xsh_synch(int nargs, char *args[])
{
    pid32   pid;
    int semStart;
    char ch;
    char *chSem;

    if(nargs == 1){
        semStart = 1;
    }else{
        chSem = args[1];
        ch = *chSem++;
        semStart = 0;
        while(ch != NULLCH) {
            if ((ch <'0') || (ch > '9')) {
                kprintf("%s: non-digit in request\n", args[1]);
                return 1;
            }
            semStart = 10*semStart + (ch - '0');
            ch = *chSem++;
        }
    }


    pid = create(synchstart, 1024, 20, "Synch_Print", 1, semStart);
    //pid = create(synchstart, 1024, 20, "Synch_Print", 0);

    resume(pid);

    return 0;

}
```

prototypes.h



```
extern  void    synchstart(int);

extern  void    runforever(void);

/* in file addargs.c */
extern  status  addargs(pid32, int32, int32[], int32,char *, void *);

/* in file ascdate.c */
extern  status  ascdate(uint32, char *);

/* in file bufinit.c */
extern  status  bufinit(void);

/* in file chprio.c */
extern  pri16   chprio(pid32, pri16);

/* in file clkupdate.S */
extern  uint32  clkcount(void);

/* in file clkhandler.c */
extern  interrupt clkhandler(void);

/* in file clkinit.c */
extern  void    clkinit(void);

/* in file clkint.S */
extern  void    clkint(void);

/* in file close.c */
extern  syscall close(did32);

/* in file control.c */
extern  syscall control(did32, int32, int32, int32);

/* in file create.c */
extern  pid32   create(void *, uint32, pri16, char *, uint32, ...);
```

shprototypes.h



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