

## תוכניות דוגמא xexec1, xexec2, xfork

התוכניות הללו נועדו להמחיש איך ברמה העקרונית ממומשים קריאות המערכת `exec1` הקיימים בכל מימוש של שפת C ו-`fork` שיטת היצור המתוחכמת של `Unix`.

קריאת המערכת `exec1` ורמץ לתהליך לנטוש את התוכנית שהוא מבצע ולבצע קוד אחר. כל תהליך מבצע בכל רגע תוכנית אחת אבל זה לא חייב להיות אותה תוכנית לאורך כל הריצה של התהליך.

עיקר הרעיון של לבצע החלפת מומחש ע"י התוכנית `xexec1` ו-`xexec2`. ההבדל בין שנתי התוכניות הללו היא ש-`xexec1` תומך בפרמטר יחיד לתוכנית החדשה ואילו `xexec2` מאפשר העברת מספר לא קבוע מראש של פרמטרים.

ההבנה איך `xexec1` עובד הוא פשוט להבין את `create`. מה ש-`create` עושה הוא להקצות כניסה בטבלת התהליכים, להקצות זיכרון למחסנית לתהליך החדש, להעביר פרמטרים מהמחסנית של תהליך האב למחסנית של התהליך החדש, להכין את התהליך החדש לסיום ולמנגנון החלפת התהליכים על ידי מילוי עבר פיקטיבי במחסנית של התהליך החדש. כאן יש לנו משימה אפילו יותר פשוטה מ-`create`. אנחנו לא צריכים להקצות כניסה חדשה בטבלת התהליכים ולא צרכים זיכרון נוסף. מה ש-`xexec1` עושה הוא פשוט עושה את החלק האחרון של `create` לעצמו: הוא ממלא את תחתית המחסנית שלו בתוכן חדש המתאים לקוד החדש: הוא רושם שם את הפרמטר היחיד, את ה-`INITRET` הסטנדרטי ורושם מעליו את הפוינטר לפונקציה לקוד החדש מעליו. מעליו הוא כותב למחסנית את התוכן המיועד למנגנון החלפת התהליכים. למעשה זה היה מה ש-`create` היה רושם אילו הקוד החדש היה הקוד ההתחלתי של התהליך. כאן כבר התהליך מוכן למנגנון החלפת התהליכים אלא שאפילו לא צריך להמתין למנגנון לשבץ את התהליך מחדש הוא פשוט קורא ל-`ctxsw` ישירות, תוך שהוא דואג שההשמה שמבצע `ctxsw` לא יעשה בעיות ע"י סיפוק כתובת של משתנה `dummy` שהוא יעד בלתי מזיק לכתיבה. אפשר לממש את אותו רעיון בצורה קצת שונה אבל העיקרון יהיה בעיקרו של דבר שכתוב המחסנית.

מימוש של `xexec2` לעומת `xexec1` הוא פשוט לממש לולאה המעתיקה מספר לא קבוע מראש של פרמטרים מהנקודה הפעילה של המחסנית לתחתית המחסנית.

קריאת המערכת `fork` ב-`Unix` היא השיטה הסטנדרטית של יצירת תהליך חדש במערכת הזו. הקריאה `fork` יוצרת כפיל של תהליך האב המבצע אותו קוד, עם עותק של שטח המשתנים של תהליך האב, מבצע אותו תוכנית וממשיך מאותו נקודה בתוכנית של אחרי החזרה מ-`fork`. חלק מהמאפיינים (כמו המספר המזהה) של שני התהליכים. כמו כן הם נבדלים בתוצאת הפונקציה: 0 לתהליך הבן וה-`pid` של תהליך הבן עבור תהליך האב.

המימוש של `xfork` הוא מורכב אבל בעיקרו של דבר המימוש שוב מבוסס על הבנת `create` ושיצירת עבר פיקטיבי לתהליך הוא דבר שניתן בקלות לעשות ע"י מילוי המחסנית. הרוטינה `xfork` מקצה את המשאבים החדשים ומתאחלת את מרכיב הכניסות

בכניסה החדשה בטבלת התהליכים תוך שהוא מעתיק מהכניסה של תהליך האב את מרבית המאפיינים. לאחר מכן הוא מעתיק מהמחסנית של תהליך האב את החלק הפעיל של המחסנית לתוך המחסנית של התהליך החדש/הבן. זה יגרום, למשל, שהמשתנים הלוקליים של התהליך החדש יהיו זהים בתוכנם לאלו של תהליך האב, וכל ביצוע של ret מהנקודה הנוכחית בקוד יחדש קוד שתהליך האב בא ממנו, דבר שיוצר אפקט של "כפיל" כביכול. התהליך החדש מועבר ע"י xfork עם יצירתו למצב PREADY.

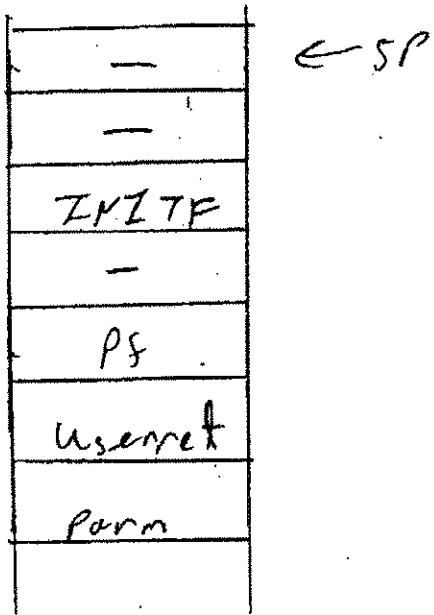
הצעד העיקרי הבא הוא להכין את התהליך החדש למנגנון החלפת התהליכים שיתחיל בעצם בנקודה קרובה לסיום xfork כאשר הוא רץ. הדבר אינו נחוץ לתהליך האב (הוא כבר שם). שוב אנחנו ממלאים תוכן מתאים ל-ctxsw כפי ש-create עושה (ללא INITRET ופרמטרים) במחסנית של תהליך הבן וכותבים את הנקודה sp לתוך השדה pregs של תהליך הבן. יש קושי טכני קטן של לגלות את ה-ip הרצוי עבור תהליך הבן. הפתרון הוא טריק סטנדרטי למדי של שימוש ב-call להסתעף לרוטינה retip הקוראת מתוך המחסנית את כתובת החזרה שלה ומחזירה אותו כתוצאת פונקציה. הדבר נעשה באסמבלי אבל אפשר להתחכם ולעשות את זה גם ב-C.

להבטיח שהמילוי הנוסף למחסנית יעשה רק ע"י תהליך האב לתהליך הבן הוא עניין פשוט לבדוק אם המספר המזהה של התהליך החוזר זהה למספר המזהה של התהליך החדש. אותה השוואה משמשת להבטיח תוצאת פונקציה שונה.

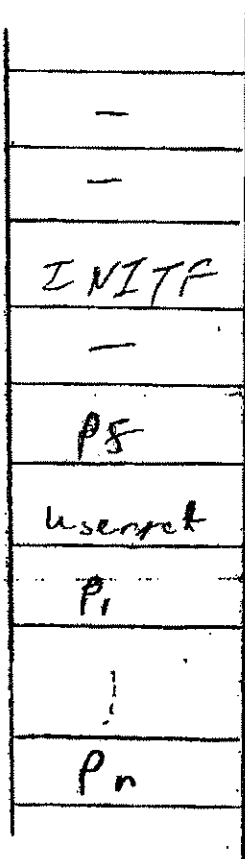
עניין טכני שאפילו fork האמיתי לא צריך להידרש (כשל המיעון הוירטואלי שקיים ב-Unix אבל לא ב-Xinu) הוא שצריך להתאם את ערכי ה-BP במחסנית של תהליך הבן כהם מצביעים למחסנית של תהליך האב ולא תהליך הבן. את זה ניתן לעשות ע"י החלפתם בחישוב ההיסט של הערכים הללו במחסנית של תהליך האב וחישוב ההיסט הזהה בתוך המחסנית של תהליך הבן. זה פותר את הבעיה של המצביעים הללו אבל לא משתני מצביע באופן כללי: כל משתנה מצביע של תהליך הבן אם הוא מצביע למשתנה לוקלי הוא יצביע למשתנה לוקלי של האב.

ב-fork האמיתי יש צורך לשכפל גם את סגמנט המידע ויש צורך לבנות טבלת המרה לוקלית נפרדת לתהליך הבן ועוד אמצעים נוספים. כמו כן fork בדרך כלל מעביר שליטה לתהליך הבן דווקא, דבר שאפשר לעשות גם ב-Xinu.

xexed 10                     



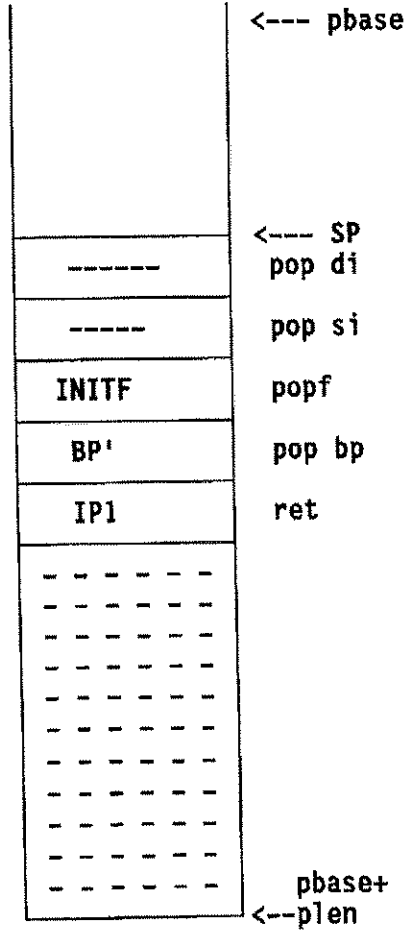
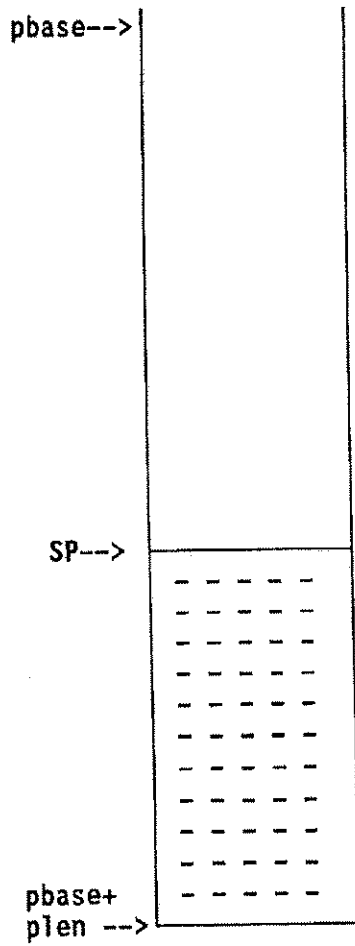
xexed2 10                     



המחסניות של fork

תהליך האב

תהליך הבן



```

/* xexecl.c - xmain, prA, prB */

#include <conf.h>
#include <kernel.h>
#include <io.h>
#include <proc.h>
#include <sem.h>
#include <mem.h>
#include <q.h>
#include <bios.h>
#include <kbdio.h>

#define INITF 0x0200

extern int INITRET();

/*-----
 * xmain -- example of creating processes in PC-Xinu
 *-----
*/

void prA(), prB();

xmain()
{
    resume( create(prA, INITSTK, INITPRIO, "proc 1", 1, 'A') );
}

/* xexecl - emulate unix execl in xinu */

xexecl(void (*pf)(), int parm)
{
    struct pentry *pptr;
    char *saddr;
    int *sp1;
    int ps;
    int dummy;

    disable(ps);
    pptr = &proctab[currpid];
    pptr->phasmsg = 0;
    sp1 = (int *) (pptr->pbase + pptr->plen);
    pptr->pargs = 1;
    * (--sp1) = parm;
    * (--sp1) = (int) INITRET;
    * (--sp1) = (int) pf;
    --sp1;
    * (--sp1) = INITF;
    sp1 -= 2;
    pptr->pregs = sp1;
    pptr->paddr = pf;

    ctxsw(&dummy, &pptr->pregs);
} /* xexecl */

```



```

/* xexecl2.c - xmain, prA, prB */

#include <conf.h>
#include <kernel.h>
#include <io.h>
#include <proc.h>
#include <sem.h>
#include <mem.h>
#include <q.h>
#include <bios.h>
#include <kbdio.h>

#define INITF 0x0200

extern int INITRET();

/*-----
 * xmain -- example of creating processes in PC-Xinu
 *-----
 */

void prA(), prchl23();

xmain()
{
    resume( create(prA, INITSTK, INITPRIO, "proc 1", 1, 'A') );
}

/* xexecl2 - emulate unix execl in xinu */

xexecl2(void (*pf)(), int n, ...)
{
    struct pentry *pptr;
    char *saddr;
    int *sp1, *a;
    int ps, i;
    int dummy;

    disable(ps);
    pptr = &proctab[currpid];
    pptr->phasmsg = 0;
    sp1 = (int *) (pptr->pbase + pptr->plen);

    pptr->pargs = n;
    a = n + 1 + &n;
    for(i=0; i < n; i++)
        * (--sp1) = * (--a);

    * (--sp1) = (int) INITRET;
    * (--sp1) = (int) pf;
    --sp1;
    * (--sp1) = INITF;
    sp1 -= 2;
    pptr->pregs = sp1;
    pptr->paddr = pf;

    ctxsw(&dummy, &pptr->pregs);
} /* xexecl2 */

```

```

/*-----
 * prA -- repeatedly print 'A' without ever terminating
 *-----
 */

void prA(int ch)
{
    int i;
    for(i=0; i< 10; i++)
    {
        putchar(CONSOLE, ch);
        putchar(CONSOLE, '\n');
    }
    sleep(5);
    xexec12(prch123, 3, 'B', 'C', 'D');
}

/*-----
 * prch123 -- repeatedly print 'ch1ch2ch3' without ever terminating
 *-----
 */

void prch123(int ch1, int ch2, int ch3)
{
    int i;
    for(i=0; i< 10; i++)
    {
        putchar(CONSOLE, ch1);
        putchar(CONSOLE, ch2);
        putchar(CONSOLE, ch3);
        putchar(CONSOLE, '\n');
    } /* for */
} /* prch123 */

```

```

A
A
A
A
A
A
A
A
A
A
BCD
BCD
BCD
BCD
BCD
BCD
BCD
BCD
BCD
BCD
BCD
BCD

```



```

/* xfork.c - xmain, prA, prB */

#include <conf.h>
#include <kernel.h>
#include <io.h>
#include <proc.h>
#include <sem.h>
#include <mem.h>
#include <q.h>
#include <bios.h>
#include <kbdio.h>

#define INITF 0x0200

extern int INITRET();

/* retip - compute ip of point of program */

int retip()
{
int ip1;

asm {
    push ax
    mov ax, [BP+2]
    mov ip1, ax
    pop ax
}
return ip1;
}

/* xfork - xinu emulation of unix fork, will work
only in the process main program, and pointers should not be used -
pointers in the child process will point into the parent variable
space */

int xfork()
{
char *saddr;
int *sp1, *sp2, *sp3, *sp4;
int ps, bpl;
int dummy;
int pid;
struct pentry *pptr, *pptr1;
int ip1;

disable(ps);
pptr = &proctab[currpid];
pid = create(pptr->paddr, pptr->plen, pptr->pprio, pptr->pname, 0);

if (pid == SYSERR)
{
restore(ps);
return SYSERR;
} /* if */

pptr1 = &proctab[pid];

asm mov sp1, sp
sp2 = pptr->pbase + pptr->plen;
sp3 = pptr1->pbase + pptr1->plen;

```

```

/* give child process a duplicate stack */
for(,sp2 >= sp1;)
{
    *sp3 = *sp2;
    sp2--;
    sp3--;
}

/* compute instruction pointer for child process */
ip1 = retip();

/* child process starts HERE */
if (curripid != pid) /* parent process only */
{
    *(int *)sp3 = ip1; /* simulate a context switch */
    sp3 -= 1;

    /* simulate call to ctksw */

    /* bp adjusting - necessary because our xinu does not support
       virtual addressing, but rather uses real addressing */

    /* bp adjusting of ctksw for child process - real mode */

    asm mov bp1, bp
    *(int *)sp3 = ((int)pptr1->pbase) + ((bp1 - ((int)pptr->pbase)));
    sp3 -= 1; /* 1 word for bp */
    *(int *)sp3 = INITF; /* FLAGS value */
    sp3 -= 1;
    sp3 -= 1; /* 2 words for si and di */

    /* complete emulation of ctksw */

    pptr1->pregs = sp3;

    /* bp adjusting of xfork for child process - real mode */
    asm mov bp1, bp
    sp4 = (int *) ( ((int)pptr1->pbase) + (( bp1 - ((int)pptr->pbase)) ));

    /* bp adjusting of xmain for child process - real mode */
    asm {
        push ax
        mov ax, [bp]
        mov bp1, ax
        pop ax
    }
    *sp4 = ( ((int)pptr1->pbase) + (( bp1 - ((int)pptr->pbase)) ));

    resume(pid);
    restore(ps);
    return pid;

} /* if */
else
    return 0; /* child process only */
} /* xfork */

```

```
/* tstxfrk.c - test xfork */
```

```
#include <conf.h>
#include <kernel.h>
#include <io.h>
#include <proc.h>
#include <sem.h>
#include <mem.h>
#include <q.h>
#include <bios.h>
#include <kbdio.h>
```

```
/*-----
 * xmain -- example of creating processes in PC-Xinu
 *-----
 */
```

```
void process()
{
    int n = 100;

    int id, *nptr;

    nptr = &n;
    if ( ( id = xfork() ) == 0 )
    { /* select child process */
        printf("\n***** child process *****\n");

        *nptr = 999; /* Only this line is different */

        printf("PID is %d and ID is %d.\n", getpid(), id);
        printf("n is %d, *nptr is %d and nptr is %d.\n", n, *nptr, nptr);
        printf("\n***** child process *****\n");

        sleep(6);
        printf("\n Press enter to continue ... ");
        getchar();

        printf("\n***** child process *****\n");
        printf("PID is %d and ID is %d.\n", getpid(), id);
        printf("n is %d, *nptr is %d and nptr is %d.\n", n, *nptr, nptr);
        printf("\n***** child process *****\n");

        n = 707;

        printf("\n***** child process *****\n");
        printf("PID is %d and ID is %d.\n", getpid(), id);
        printf("n is %d, *nptr is %d and nptr is %d.\n", n, *nptr, nptr);
        printf("\n***** child process *****\n");

        return;
    }
}
```

```

sleep(5);
printf("\n***** parent process *****\n");
printf("PID is %d and ID is %d.\n", getpid(), id);
printf("n is %d, *nptr is %d and nptr is %d.\n", n, *nptr, nptr);
printf("\n***** parent process *****\n");

n = 200;
*nptr = 300;
printf("\n***** parent process *****\n");
printf("PID is %d and ID is %d.\n", getpid(), id);
printf("n is %d, *nptr is %d and nptr is %d.\n", n, *nptr, nptr);
printf("\n***** parent process *****\n");
while(1)
    ;

} /* process */

xmain()
{
resume(create(process, INITSTK, INITPRIO, "process", 0));
} /* xmain */

```

E:\USERS\EYTAN\XINU4WIN\NEWSRC\EXAMPLES>tstxfrk  
Initializing . . .

PC-Xinu Version 6pc (1-Dec-87)  
63864 real mem  
18312 base addr  
45552 avail mem

Hit any key to continue . . .

\*\*\*\*\* child process \*\*\*\*\*  
PID is 24 and ID is 0.  
n is 100, \*nptr is 999 and nptr is 26138.

\*\*\*\*\* child process \*\*\*\*\*

\*\*\*\*\* parent process \*\*\*\*\*  
PID is 25 and ID is 24.  
n is 999, \*nptr is 999 and nptr is 26138.

\*\*\*\*\* parent process \*\*\*\*\*

\*\*\*\*\* parent process \*\*\*\*\*  
PID is 25 and ID is 24.  
n is 300, \*nptr is 300 and nptr is 26138.

\*\*\*\*\* parent process \*\*\*\*\*

Press enter to continue ...

\*\*\*\*\* child process \*\*\*\*\*  
PID is 24 and ID is 0.  
n is 100, \*nptr is 300 and nptr is 26138.

\*\*\*\*\* child process \*\*\*\*\*

\*\*\*\*\* child process \*\*\*\*\*  
PID is 24 and ID is 0.  
n is 707, \*nptr is 300 and nptr is 26138.

\*\*\*\*\* child process \*\*\*\*\*

# דגמאות נוספות

```
/* async1.c - prod2, cons2 */
```

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
```

```
volatile int    *n;
```

```
/*-----
 * main -- producer and consumer processes synchronized with semaphores
 *-----
 */
```

```
int main()
{
    int    prod2(), cons2();
    int    semid, shmid;
    struct shmids buff;

    shmid = shmget(0, sizeof(int), 0666);
    n = (int*)shmat(shmid, 0, 0);
    *n = 0;

    if ( fork() )
        /* Parent process */
        cons2();
    else /* Child process */
        prod2();

    shmdt((char *)n);

    shmctl(shmid, IPC_RMID, &buff);

    return 0;
}
```

```
/*-----
 * prod2 -- increment n 20 times
 *-----
 */
```

```
int prod2()
{
    int    i;

    for (i=1; i<=20; i++) {
        (*n)++;
    }
}
```

```
/*-----
 * cons2 -- print n 20 times
```

222





```

/* async2.c - prod2, cons2 */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>

volatile int    *n;

/*-----
 * main -- producer and consumer processes synchronized with semaphores
 *-----
 */
int main()
{
    int    prod2(), cons2();
    int    semid, shmid;
    struct shmids buff;

    shmid = shmget(IPC_PRIVATE, sizeof(int), 0666);
    n = (int*)shmat(shmid, 0, 0);
    *n = 0;

    if ( fork() == 0 )
        /* Child process */
        cons2();
    else /* Parent process */
        prod2();

    shmdt((char *)n);
    shmctl(shmid, IPC_RMID, &buff);

    return 0;
}

/*-----
 * prod2 -- increment n 20 times
 *-----
 */
int prod2()
{
    int    i;

    for (i=1; i<=20; i++) {
        (*n)++;
    }
}

/*-----
 * cons2 -- print n 20 times
 *-----

```



```

/* demo_sig_par.c -- traps a signal - */

#include <stdio.h>
#include <signal.h>

void handler(int signum)
{
    if(signum == SIGINT)
    {
        signal(SIGINT, handler);
        printf("I wont be interrupted!\n");
    } /* if */
    else
        if(signum == SIGTSTP)
        {
            signal(SIGTSTP, handler);
            printf("I wont be suspended!\n");
        } /* if */
} // handler

int main()
{
    int i, pid;

    signal(SIGINT, handler);
    signal(SIGTSTP, handler);

    for (i=1; i < 50; i++)
    {
        printf("Ha Ha Ha Ha \n");
        sleep(2);
    } /* for */

    return 0;
} /* main */

```

---

```

% cc demo_sig_par.c
% ./a.out
Ha Ha Ha Ha
Ha Ha Ha Ha
Ha Ha Ha Ha
I wont be suspended!
Ha Ha Ha Ha
Ha Ha Ha Ha
I wont be interrupted!
Ha Ha Ha Ha
Ha Ha Ha Ha
Quit
%

```

```

// uncrt3.c

#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <sys/types.h>

int create(char path[], char argv0[], char parm1[], char parm2[],
char parm3[])
{
    int pid;

    if ( (pid = fork()) == 0 )
    {
        kill(getpid(), SIGSTOP);
        execl(path, argv0, parm1, parm2, parm3, 0 );
        exit(0);
    } /* child */
    else /* parent */
        return pid;
} /* create */

int resume(int pid)
{
    return (kill(pid, SIGCONT));
} /* resume */

int main(void)
{
    int pid;

    printf("Before Create\n");
    pid = create("/bin/ping", "ping", "-c", "2", "study.haifa.ac.il");
    sleep(3);
    printf("Before Resume\n");
    resume(pid);

    return 0;
} /* main */

```

---

```

% cc uncrt3.c
% ./a.out
Before Create
Before Resume
% PING study.haifa.ac.il (132.74.1.26) 56(84) bytes of data.

```

227

64 bytes from study.haifa.ac.il (132.74.1.26): icmp\_seq=1 ttl=254  
time=0.445 ms

64 bytes from study.haifa.ac.il (132.74.1.26): icmp\_seq=2 ttl=254  
time=0.390 ms

--- study.haifa.ac.il ping statistics ---

2 ~~packets transmitted~~, 2 received, 0% packet loss, time 999ms

rtt min/avg/max/mdev = 0.390/0.417/0.445/0.034 ms

%

228

```

/* sync1.c - prod2, cons2 */

#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

typedef struct n
{
    int n;
    int flag1;
    int flag2;
} N, *NPTR;

volatile NPTR n;

/*-----
 * main -- producer and consumer processes synchronized with semaphores
 *-----
 */
int main()
{
    int prod2(), cons2();
    int shmids;
    struct shmids buff;

    shmids = shmget(IPC_PRIVATE, sizeof(N), 0666);
    n = (NPTR)shmat(shmids, 0, 0);
    n->n = 0;
    n->flag1 = 0;
    n->flag2 = 1;

    if ( fork() )
        /* Parent process */
        cons2();
    else /* Child process */
        prod2();

    shmdt((char *)n);
    shmctl(shmids, IPC_RMID, &buff);

    return 0;
}

/*-----
 * prod2 -- increment n 20 times
 *-----
 */

```

229

```
int prod2()
{
    int    i;

    for (i=1; i<=20; i++) {
        while(n->flag1 == 0)
            (n->n)++;
        n->flag2 = 1;
        n->flag1 = 0;
    }
    return 0;
}
```

```
/*-----
 * cons2 -- print n 20 times
 *-----
*/
```

```
int cons2()
{
    int    i;

    for (i=1; i<=20; i++) {
        while(n->flag2 == 0)
            printf("n is %d\n", n->n);
        n->flag1 = 1;
        n->flag2 = 0;
    }
    return 0;
}
```

```
% cc sync1.c
% a.out
n is 0
n is 1
n is 2
n is 3
n is 4
n is 5
n is 6
n is 7
n is 8
n is 9
n is 10
n is 11
n is 12
n is 13
n is 14
n is 15
```

n is 16  
n is 17  
n is 18  
n is 19  
%

---

231