מערכות הפעלה תרגול 6

Signals – Race & Masking

מתרגל-יורם סגל
yoramse@post.bgu.ac.il

T7 Contents

סיגנל בתוך סיגנל

- מרוצים

סינכרון סיגנלים

- sigprocmask()

מרוץ כנגד sigprocmask()

- מנגנון בטוח של POSIX
- sigaction()

t5_1.c - תוך סיגנל

hello world SIGUSR1

int glob=1;

Usr1.txt: the value of glob is 2

SIGUSR1=> Defualt

SIGUSR2=> Defualt

int glob=0;

```
int main (void
pid_t child_id;
int status;
signal(SIGUSR1,signal_hand);
signa
child_id = fork (); /*child process*/
if (child_id > 0){ //father
   kill(child_id,SIGUSR1);
   kill(child_id,SIGUSR2);
   wait(&status);
   printf("father is done\n");}
if (child_id == 0){
   while(glob<2);
      printf("son is done\n");}
return 0;}
```

hello world SIGUSR 2

int glob=2;

Usr2.txt: the value of glob is 2

SIGUSR1=> signal_hand

SIGUSR2=> signal_hand2

WAIT-האב ימתין עד שהמצב (state) ישתנה.

father is done

son is done

האבא שלח סיגנל 2 לילד. הילד הפסיק פעולת פונקציה זו והתחיל להתקדם בפונקציה של סיגנל 2

```
void signal_hand (int sig){
int i,fd;
signal(SIGUSR1,signal_hand);
printf("hello world SIGUSR1 %d\n",sig);
glob+=1;
for(i=0;i<1000000000;i++); //wait
printf("the value of glob is %d\n",glob);
fd=open("usr1.txt",O_.....);
write(fd,"hello world",10);
close(fd);}
```

הילד סיים לרוץ בתוך הפונקציה של סיגנל 2 ולכן חוזר להמשיך מהמקום שהפסיק בפונקציה של סיגנל 1

```
void signal_hand2 (int sig){
int i,fd;
signal(SIGUSR2,signal_hand2);
printf("hello world SIGUSR2 %d\n",sig);
glob+=2;
for(i=0;i<1000000000;i++); //wait
printf("the value of glob is %d\n",glob);
fd=open("usr2.txt",O_.....);
write(fd,"hello world",10);
close(fd);}
```

האבא שלח סיגנל (KILL לא יכול לשלוח סיגנל לעצמו ולכן הסיגנל מגיע רק לילד). הילד מפסיק לרוץ, במקום שהסיגנל תפס אותו, ובמקום זאת, הילד מתקדם בתוך הפונקציה של הסיגנל

דוגמה שניה signal races

```
float A=0;
int main (void){
pid_t child_id;
int status;
signal(SIGUSR1,signal_hand1);
signal(SIGUSR2,signal_hand2);
while();
```

```
void signal_hand1 (int sig){
int i,fd;
signal(SIGUSR1,signal_hand);
If A<=0 {
A=A+1;
A=3*A;
printf(A1=%f\n, A);
}
```

Expected

3 , 1

A=0	A=0
A=A+1 ➔ A=1	A=A+1 ➔ A=1
A=3*A; ➔ A=3	A=A-1; ➔ A=0
Print A1=3	A=A/2; ➔ A=0
A=A-1; ➔ A=2	print A2=0
A=A/2; ➔ A=1	Print A1=3
print A2=1	

In practice might be:

0 , 3 or 3, 1 or …

```
void signal_hand2 (int sig){
int i,fd;
signal(SIGUSR2,signal_hand);
If A>0 {
A=A-1;
A=A/2;
printf(A2=%f\n, A);
}
```

פתרון בעיית מרוצים - סנכרון/חסימה

- פתרון: ניתן לגרום לתהליך לחסום סיגנלים מסוימים במהלך הריצה שלו.
  - ברגע שנפעיל את החסימה כל הסיגנלים מהסוג שאותו חסמנו, לא יתקבלו במהלך ריצת התהליך עד שנסיר את החסימה.
  - הסיגנל החסום יופעל לאחר שהחסימה תוסר.
  - יש סיגנלים שלא ניתן לחסום: SIGKILL, SIGSTOP.

חסימת סיגנלים - sigprocmask

- אז כמו שכבר אמרנו -
  - הפונקציה signal משייכת פונקציה לסיגנל. בכל פעם שמקבלים סיגנל, חייבים להגדיר מחדש את השיוך.
  - היא לא מונעת מצב של signal races

- ולכן נשתמש בפונקציה **sigprocmask**
  - מגדירה איזה סיגנלים לחסום עבור התהליך (במהלך הריצה התקינה שלו).
  - ניתן להשתמש בפונקציה בתוך signal handler על מנת לחסום/לשחרר סיגנלים.

שיטה 1: מניעת סיגנל בתוך סיגנל פשוטה אך לא מושלמת

הגדרה בשני שלבים:

1. signal() הצמדת פונק' לסיגנל
2. sigprocmask() - הגדרת מיסוך בתוך פונק' הסיגנל

חסימת סיגנלים
sigprocmask(…sigset_t…)

הגדרת איתותים לחסימה
sigset_t

פונקציות למניפולציה בדגלי חסימה

- int sigemptyset(sigset_t *mask);
  Clears the mask (no signals are flagged as blocked)
- int sigfillset(sigset_t * mask);
  All signals are flagged as blocked
- int sigaddset(sigset_t * mask, int signum);
  Adds a signum to the blocked signals
- int sigdelset(sigset_t * mask, int signum);
  Removes a signum from the blocked signals
- int sigismember(sigset_t * mask, int signum);
  Returns 0/1 if a particular signal is flagged as blocked

שימוש: בתחילת ההנדלר אני מפעיל את החסימה
חיסרון: מרגע שקוראים להנדלר ועד הפעלת החסימה יכול ל"התפלח" סיגנל אחר (מרוץ)

שיטה 2: הגדרה משולבת של פונקציה לטיפול + איתותים לחסימה

הגדרה משולבת של פונקציה לטיפול+איתותים לחסימה
sigaction(… sigaction…)

sa_flags
שדה sa_flagsמורכב ממספר דגלים של ביט המופרדים ע"י OR לדוגמא:

SA_RESETHAND: מחזיר את ה handler לפעולה הדיפולטיבית.
SA_NODEFER: מתעלם מהשדה sa_mask (בעייתי מבחינת סינכרון סיגנלים).
SA_SIGINFO: אל תשתמש בפונקציית ה handler שמוגדרת בשדה sa_handler,
במקום זה תשתמש בפונקציה שמוגדרת בשדה *sa_sigaction*

```
struct sigaction {
   void (*sa_handler)(int);
   sigset_t sa_mask;
   int
   voi
};
```

POSIX extension

```
siginfo_t {
        int si_signo; /* Signal number */
        int si_errno; /* An errno value */
        int si_code; /* Signal code */
        int si_trapno; /* Trap number that caused hardware-generated signal (unused on most architectures) */
        pid_t si_pid; /* Sending process ID */
        uid_t si_uid; /* Real user ID of sending process */
        int si_status; /* Exit value or signal */
        clock_t si_utime; /* User time consumed */
        clock_t si_stime; /* System time consumed */
        sigval_t si_value; /* Signal value */
        int si_int; /* POSIX.1b signal */
        void *si_ptr; /* POSIX.1b signal */
        int si_overrun; /* Timer overrun count; POSIX.1b timers */
        int si_timerid; /* Timer ID; POSIX.1b timers */
        void *si_addr; /* Memory location which caused fault */
        long si_band; /* Band event (was int in glibc 2.3.2 and earlier) */
        int si_fd; /* File descriptor */
        short si_addr_lsb; /* Least significant bit of address (since kernel 2.6.32) */ }
```

sa_sigaction:
This function receives the:
signal number as its first argument.
a pointer to a siginfo_t as its second argument.

שיטה 1: חסימת איתותים כללית

#include <signal.h> פקודה לטעינת מילת חסימת סיגנלים (מסכה)

int sigprocmask(int *how*, const sigset_t **set*, sigset_t **oldset*);

```
                  ---------------------------------
signal masks      |  |  |  . . |0 | . .   |  |  | 0=no BLOCK, 1=BLOCK
                  ---------------------------------
```

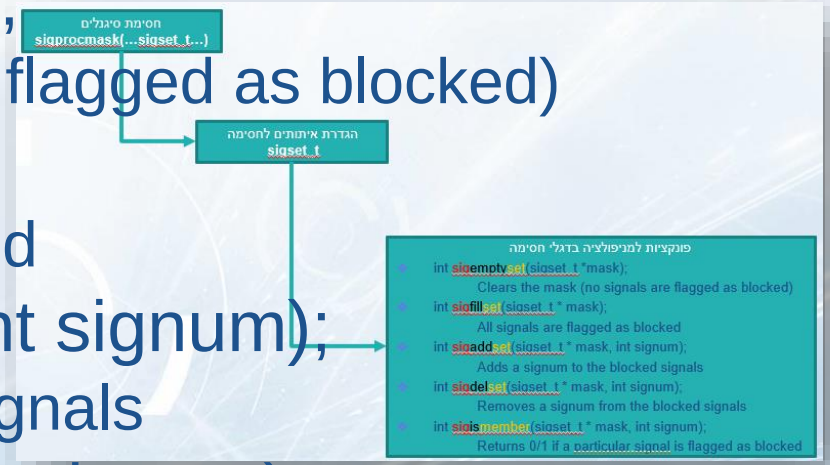
המסכה מסוג **sigset_t** היא מילה בינארית אשר כל ביט בה מייצג סיגנל אחר.
אם הביט הוא 1 אזי הסיגנל חסום.
בשביל לא להתעסק עם אופרטרים בינאריים and, or, xor
יש לנו פונקציות מוכנות שיודעות להוסיף ולמחוק סיגנלים מתוך המסכה

חסימת איתותים כללית

- פרמטר how:
  - **SIG_BLOCK – במסכה מוגדרים רק האיתותים הנוספים שצריך לחסום (ADD).**
  - **SIG_UNBLOCK - במסכה מוגדרים רק האיתותים הנוספים שצריך להוריד את חסימתם (DELETE).**
  - **SIG_SETMASK – המסכה הופכת להיות המסכה החדשה של התהליך (SET).**
- פרמטר const sigset_t *set: סט הסיגנלים.
- פרמטר sigset_t *oldset: אם לא NULL, מחזיר את ה-mask הקודם.

Create the Signal Masks

- sigset_t sa_mask:
  - איזה איתותים לחסום כאשר מתבצע ה handler.
- int sigemptyset(sigset_t *mask);
  - Clears the mask (no signals are flagged as blocked)
- int sigfillset(sigset_t * mask);
  - All signals are flagged as blocked
- int sigaddset(sigset_t * mask, int signum);
  - Adds a signum to the blocked signals
- int sigdelset(sigset_t * mask, int signum);
  - Removes a signum from the blocked signals
- int sigismember(sigset_t * mask, int signum);
  - Returns 0/1 if a particular signal is flagged as blocked



SIGPROCMASK examples

- `sigset_t set;`

- `sigemptyset(&set);` //Clears the mask (no signals are flagged as blocked)
- `sigaddset(&set, SIGINT);` //Adds `SIGINT` to the blocked signals
- `sigaddset(&set, SIGTERM);` //Adds `SIGTERM` to the blocked signals
- `sigprocmask(SIG_SETMASK, &set, NULL);` //blocked **only**: SIGINT and SIGTERM

- `sigemptyset(&set);`
- `sigaddset(&set, SIGINT);`
- `sigaddset(&set, SIGALRM);`
- `sigprocmask(SIG_BLOCK, &set, NULL);`//**Add** blocked signals: SIGINT, SIGALRM

- `sigemptyset(&set);`
- `sigaddset(&set, SIGTERM);`
- `sigaddset(&set, SIGUSR1);`
- `sigprocmask(SIG_UNBLOCK, &set, NULL);`//unblock signals: SIGTERM and SIGUSR1

SIG_BLOCK – במסכה מוגדרים רק האיתותים הנוספים שצריך לחסום (ADD).
SIG_UNBLOCK - במסכה מוגדרים רק האיתותים הנוספים שצריך להוריד את חסימתם (DELETE).
SIG_SETMASK – המסכה הופכת להיות המסכה החדשה של התהליך (SET).

SIGPROCMASK example

SIGTERM מספק דרך אלגנטית לסגור (להרוג) תוכנית.
זו הדרך המנומסת להרוג אפליקציה או תוכנית
התנהגות ברירת המחדל של פקודת kill היא שליחת אות SIGTERM לתהליך.

- Let's Blocked signals: {SIGSEGV, SIGTERM}

segmentation violation

```
sigset_t x, y;
sigemptyset (&x);
sigaddset(&x, SIGUSR1);
sigprocmask(SIG_BLOCK, &x, &y)
```

New mask

x == {SIGUSR1}

SIG_BLOCK - במסכה מוגדרים רק
האיתותים הנוספים שצריך לחסום (ADD)

Old mask

y == {SIGSEGV, SIGTERM}

חסרונות sigprocmask

- השימוש ב ()sigprocmask לא פותר את כל בעיות סנכרון הסיגנלים.
- לדוגמא, יכול לקרות מצב שבו אנחנו כבר נמצאים ב signal handler וסיגנל נוסף הגיע לפני שהספקנו לקרוא ל ()sigprocmask.
- על מנת להבטיח שלא יהיו בעיות סנכרון, נרצה להשתמש במנגנון מובנה יותר שמבטיח שכאשר נטפל בסיגנל אף סיגנל אחר לא יפריע.

הדרך למנוע מרוצים, היא לאפשר, למערכת ההפעלה, לקבוע את מיסוך הסיגנל עבורנו, לפני שהיא תממשק (תחבר) בין הסיגנל לבין הפונקציה שמטפלת בסיגנל (signal handler).
ניתן לעשות זאת אם אנו משתמשים ב()sigaction כדי להגדיר בו זמנית הן את הפונקציה של הסגנל והן את מיסוך הסגינל (מסיכת האיתותים הדרושה בזמן הפעלת פונקציית האיתות.

sigaction() - POSIX handler

- נשתמש במנגנון בטוח של POSIX.
- הגרסה של POSIX ל signal handling יותר מורכבת אבל מונעת המון בעיות סנכרון.
- כלומר, הפונקציה signal() לא מונעת מסיגנל חדש להגיע במהלך הטיפול בסיגנל הנוכחי.
- לעומת זאת , הפונקציה **sigaction()** דואגת לחסימה של סיגנלים מסוימים (על פי הגדרתנו) במהלך הטיפול.

Sigaction system call

- **#include <signal.h>**
- **int sigaction(int *signum*, const struct sigaction **act*, struct sigaction **oldact*);**

גם מגדירים פונקציית תגובה לסיגנל וגם מגדירים מיסוכי סיגנלים והכל בשלב ההגדרה

```
struct sigaction {
    void (*sa_handler)(int);
    sigset_t sa_mask;
    int sa_flags;
    void (*sa_sigaction)(int, siginfo_t *, void *);
};
```

פונקציית תגובה

מיסוך סיגנלים

specifies the action to be associated with signum

השדה השני מכיל מבנה שיכיל אינפורמציה על האיתות שנשלח

If the flag **SA_SIGINFO** is specified in **sa_flags**, then **sa_sigaction** (instead of **sa_handler**) specifies the signal-handling function for signum.

int sigaction(int *signum*, const struct sigaction **act*, struct sigaction **oldact*);

```
struct sigaction {
   void (*sa_handler)(int);
   sigset_t sa_mask;
   int sa_flags;
   void (*sa_sigaction)(int, siginfo_t *, void *);
};
```

השדה השני מכיל מבנה שיכיל אינפורמציה על האיתות שנשלח

void (***sa_sigaction**)(int, siginfo_t *, void *);

- **sa_sigaction:**
  - This function receives the:
    - **signal number** as its first argume
    - **a pointer to a siginfo_t**
    - **a pointer to a ucontext_t** (cast t (Commonly, the handler function argument. See getcontext(3) for ucontext_t.)

```
siginfo_t {
    int si_signo; /* Signal number */
    int si_errno; /* An errno value */
    int si_code; /* Signal code */
    int si_trapno; /* Trap number that caused hardware-generated signal (unuse
    pid_t si_pid; /* Sending process ID */
    uid_t si_uid; /* Real user ID of sending process */
    int si_status; /* Exit value or signal */
    clock_t si_utime; /* User time consumed */
    clock_t si_stime; /* System time consumed */
    sigval_t si_value; /* Signal value */
    int si_int; /* POSIX.1b signal */
    void *si_ptr; /* POSIX.1b signal */
    int si_overrun; /* Timer overrun count; POSIX.1b timers */
    int si_timerid; /* Timer ID; POSIX.1b timers */
    void *si_addr; /* Memory location which caused fault */
    long si_band; /* Band event (was int in glibc 2.3.2 and earlier) */
    int si_fd; /* File descriptor */
    short si_addr_lsb; /* Least significant bit of address (since kernel 2.6.32) */ }
```

siginfo_t

- השדה השני מכיל מבנה שיכיל אינפורמציה על האיתות שנשלח:

```
siginfo_t {
    int si_signo; /* Signal number */
    int si_errno; /* An errno value */
    int si_code; /* Signal code */
    int si_trapno; /* Trap number that caused hardware-generated signal (unused on most architectures) */
    pid_t si_pid; /* Sending process ID */
    uid_t si_uid; /* Real user ID of sending process */
    int si_status; /* Exit value or signal */
    clock_t si_utime; /* User time consumed */
    clock_t si_stime; /* System time consumed */
    sigval_t si_value; /* Signal value */
    int si_int; /* POSIX.1b signal */
    void *si_ptr; /* POSIX.1b signal */
    int si_overrun; /* Timer overrun count; POSIX.1b timers */
    int si_timerid; /* Timer ID; POSIX.1b timers */
    void *si_addr; /* Memory location which caused fault */
    long si_band; /* Band event (was int in glibc 2.3.2 and earlier) */
    int si_fd; /* File descriptor */
    short si_addr_lsb; /* Least significant bit of address (since kernel 2.6.32) */ }
```

- השדות משתנים כתלות בסיגנל. למשל –
  - SIGSEGV יעדכן את השדה si_addr עם הכתובת של ה fault. לעומת SIGUSR1 שיתן לו ערך NULL

sa_mask

- **sa_mask** specifies a mask of signals which should be blocked during execution of the signal handler.

- In addition, the signal which triggered the handler will be blocked, unless the SA_NODEFER flag is used.

int sigaction(int *signum*, const struct sigaction ***act*, struct sigaction ***oldact*);

```
struct sigaction {
   void (*sa_handler)(int);
   sigset_t sa_mask;
   int sa_flags;
   void (*sa_sigaction)(int, siginfo_t *, void *);
};
```

השדה השני מכיל מבנה שיכיל אינפורמציה על האיתות שנשלח

sa_flags

- שדה **sa_flags** מורכב ממספר דגלים של ביט המופרדים ע"י OR, לדוגמא:
  - SA_RESETHAND: מחזיר את ה handler לפעולה הדיפולטיבית.
  - SA_NODEFER: מתעלם מהשדה sa_mask (בעייתי מבחינת סינכרון סיגנלים).
  - SA_SIGINFO: אל תשתמש בפונקציית ה handler שמוגדרת בשדה sa_handler, במקום זה תשתמש בפונקציה שמוגדרת בשדה *sa_sigaction*

```
int sigaction(int signum, const struct sigaction *act, struct sigaction *oldact);
```

```
struct sigaction {
   void (*sa_handler)(int);
   sigset_t sa_mask;
   int sa_flags;
   void (*sa_sigaction)(int, siginfo_t *, void *);
};
```

השדה השני מכיל מבנה שיכיל אינפורמציה על האיתות שנשלח

Sigaction example t5_2.c

נחזור על תרגיל 1
הפעם נבצע זאת בצורה מוגנת
תוך התבסות על פונקציית
Sigaction() של POSIX

```
struct sigaction usr_action;
sigset_t block_mask;

/* Establish the signal handler. */
  sigfillset (&block_mask);
  usr_action.sa_handler = signal_hand;
  usr_action.sa_mask = block_mask;
  usr_action.sa_flags = 0;
sigaction (SIGUSR1, &usr_action, NULL);
```

השקפים הבאים הם למטרת תרגול עצמי למעוניינים

sa_sigaction example t5_3.c

- דגשים בתרגיל

```
sigfillset (&block_mask);
usr_action.sa_mask = block_mask;
act.sa_sigaction = sighandler;
act.sa_flags = SA_SIGINFO;
sigaction(SIGTERM, &act, NULL);
```

SIGTERM	15	Term	Termination signal

If the flag **SA_SIGINFO** is specified in **sa_flags**, then **sa_sigaction** (instead of **sa_handler**) specifies the signal-handling function for signum.

sa_sigaction example t5_3.c

```
#include <stdio.h>
#include <signal.h>
#include <string.h>
#include <unistd.h>
```

```
void sighandler(int signum, siginfo_t *info, void *ptr){
printf("Received signal %d\n", signum);
printf("Signal originates from process %lu\n",(unsigned long)info->si_pid);}
```

```
int main(){
struct sigaction act;
sigset_t block_mask;
printf("I am %lu \n", (unsigned long)getpid());
sigfillset (&block_mask);
act.sa_mask = block_mask;
act.sa_flags = SA_SIGINFO;
act.sa_sigaction = sighandler;
sigaction(SIGTERM, &act, NULL);
sleep(60);
printf("done\n");
return 0;}
```

1. Compile t5_3.c
2. Run t5_3.out
3. Use ps to find your PID
4. Send the command **kill 2563**
   It would send a signal called SIGTERM to the process.
   You should do it in less than 60 sec.

בדיקת איתותים חסומים הממתינים

- כדי לבדוק האם יש איתותים חסומים שממתינים להורדת החסימה כדי להתבצע:
- **int sigpending(sigset_t *set);**

הפונקציה מקבלת set ריק וממלאת את ה set באיתותים שממתינים.

איך בודקים האם איתות מסויים קיים ב set?

בעזרת הפונקציה sigismember()

בדיקת איתותים חסומים הממתינים

- האם הילד יורש את הסיגנלים הממתינים לתהליך האבא?
  - לא! האבא הוא התהליך היחיד שיצטרך להתמודד עם הסיגנלים הממתינים לאחר הורדת החסימה.
- האם exec יורש את הסיגנלים הממתינים לתהליך המקורי?
  - כן, אפילו שהוא מריץ תוכנית אחרת הוא עדיין אותו תהליך (עם אותו pid) ולכן הסיגנלים עדיין ממתינים לו.
- על מנת להתעלם מהסיגנלים אחרי הורדת החסימה ניתן להגדיר להם את פעולת הIGN (ראינו כבר איך עושים את זה)

```
signal(SIGINT, SIG_IGN);
```

```
act.sa_flags = SA_SIGINFO;
act.sa_sigaction = SIG_IGN;
```

דוגמא t5_4.c, דוגמא כוללת.

```c
#include <signal.h>
#include <unistd.h>
#include <stdio.h>

void catcher( int sig ) {
    puts( "inside catcher() function\n" );
}

void check_pending( int sig, char *signame ) {

    sigset_t sigset;

    if( sigpending( &sigset ) != 0 )
        perror( "sigpending() error\n" );

    else if( sigismember( &sigset, sig ) )
            printf( "a %s signal is pending\n", signame );
        else
            printf( "no %s signals are pending\n", signame );
}
```

```c
int main( int argc, char *argv[] ) {

    struct sigaction sigact;
    sigset_t sigset;

//do not block a signal
    //sigemptyset( &sigact.sa_mask );
sigfillset( &sigact.sa_mask );
    sigact.sa_flags = 0;
    sigact.sa_handler = catcher;

    if( sigaction( SIGUSR1, &sigact, NULL ) != 0 )
        perror( "sigaction() error\n" );

    else {
        sigemptyset( &sigset );
        sigaddset( &sigset, SIGUSR1 );
        if ( sigprocmask( SIG_SETMASK, &sigset, NULL ) != 0)
            perror( "sigprocmask() error\n" );

        else {
            printf( "SIGUSR1 signals are now blocked\n" );

            kill( getpid(), SIGUSR1 );
            printf( "after kill()\n" );

            check_pending( SIGUSR1, "SIGUSR1" );

            sigemptyset( &sigset );
            sigprocmask( SIG_SETMASK, &sigset, NULL );
            printf( "SIGUSR1 signals are no longer blocked\n" );

            check_pending( SIGUSR1, "SIGUSR1" );
        }
    }
    return( 0 );
}
```

דוגמא ל חסימה/הסרת איתותים כללית

- דוגמא t5_4.c, דוגמא כוללת.
- פלט:

```
SIGUSR1 signals are now blocked
after kill()
a SIGUSR1 signal is pending
inside catcher() function
SIGUSR1 signals are no longer blocked
no SIGUSR1 signals are pending
```

Process ID

- **pid_t getpid()**
  - **Returns:** process ID of calling process.
- **pid_t getppid()**
  - **Returns:** parent process ID of calling process.
- **uid_t getuid()**
  - **Returns:** user ID of calling process.
- **uid_t getgid()**
  - **Returns:** group ID of calling process.

Process ID – t5_5.c

- **pid_t getpgid(pid_t pid);**
  Returns: the process group ID for a process.
- **int setpgid(pid_t pid, pid_t pgid);**
  - **sets the PGID (***Process Group ID)* **of the process specified by pid to pgid.**
  - **If pid is zero, then the process ID of the calling process is used. If pgid is zero, then the PGID of the process specified by pid is made the same as its process ID.**

```c
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <unistd.h>

// This function will handle a signal.
void HandleSignal(int sig, siginfo_t *si, void *context);
```

```c
void HandleSignal(int sig, siginfo_t *si, void *context)
{
  printf("\nControl-C was pressed: mypid = %d, mypgid = %d\n",
  getpid(), getpgid(getpid()));
  _exit(0);

}
```

t5_5.c

```
planet hershks1 118 : a.out

My process id = 24027.
My process group id = 24027.

Child: My process id = 24028.
Child: My process group id = 24027.

Child: My process id = 24028.
Child: My process group id = 24028.

Parent: My process id = 24027.
Parent: My process group id = 24027.

Control-C was pressed: mypid = 24027, mypgid = 24027
planet hershks1 119 :
```

```c
int main(int argc, char *argv[])
{
    struct sigaction sVal;
    pid_t myPID;
    pid_t myG_PID;

    // Specify that we will use a signal handler that takes three arguments
    // instead of one, which is the default.
    sVal.sa_flags = SA_SIGINFO;

    // Indicate which function is the signal handler.
    sVal.sa_sigaction = HandleSignal;

    myPID = getpid();
    myG_PID = getpgid(myPID);
    printf("\nMy process id = %d.\n", myPID);
    printf("My process group id = %d.\n", myG_PID);

    if(fork() == 0)
    {
        myPID = getpid();
        myG_PID = getpgid(myPID);
        printf("\nChild: My process id = %d.\n", myPID);
        printf("Child: My process group id = %d.\n", myG_PID);

        // Create a new process group that contains this process
        setpgid(0,0);

        myPID = getpid();
        myG_PID = getpgid(myPID);
        printf("\nChild: My process id = %d.\n", myPID);
        printf("Child: My process group id = %d.\n", myG_PID);
    }
    else
    {
        // Register for SIGINT
        sigaction(SIGINT, &sVal, NULL);

        myPID = getpid();
        myG_PID = getpgid(myPID);
        printf("\nParent: My process id = %d.\n", myPID);
        printf("Parent: My process group id = %d.\n", myG_PID);

        while(1);
    }

    return(0);
}
```