setsid(), getpgrp(), getpgid(), & setpgrp(), setpgid(), kill()

Introduction

- In UNIX-like operating systems, a **process group** denotes a collection of one or more processes.
- It is used to control distribution of a signal i.e. when a signal is directed to a process group, each member of the process group receives the signal.
- Collection of processes associated with same job/terminal.
- Each process group has a unique process group id.
- Process group ids are positive integers and can be stored in pid_t data type.
- Each process group can have a process group leader.
  - Leader is identified by its pgid==pid
  - Leader can create a new process group ,create processes in the group

Introduction

Sessions

- Collection of one or more process groups.
- If the calling process is not a process group leader,this function creates a new session. Three things happen:
  - the process becomes the session leader of this new session.
  - the process becomes the process group leader of a new process group.
  - The process has no controlling terminal.

setsid()

- **Syntax**

  ```
  pid_t  setsid(void)
  ```

- **Description**

  Creates a new session with the calling process as its session leader. The caller becomes the process group leader of a new process group. The calling process must not be a process group leader already. The caller does not have a controlling terminal.

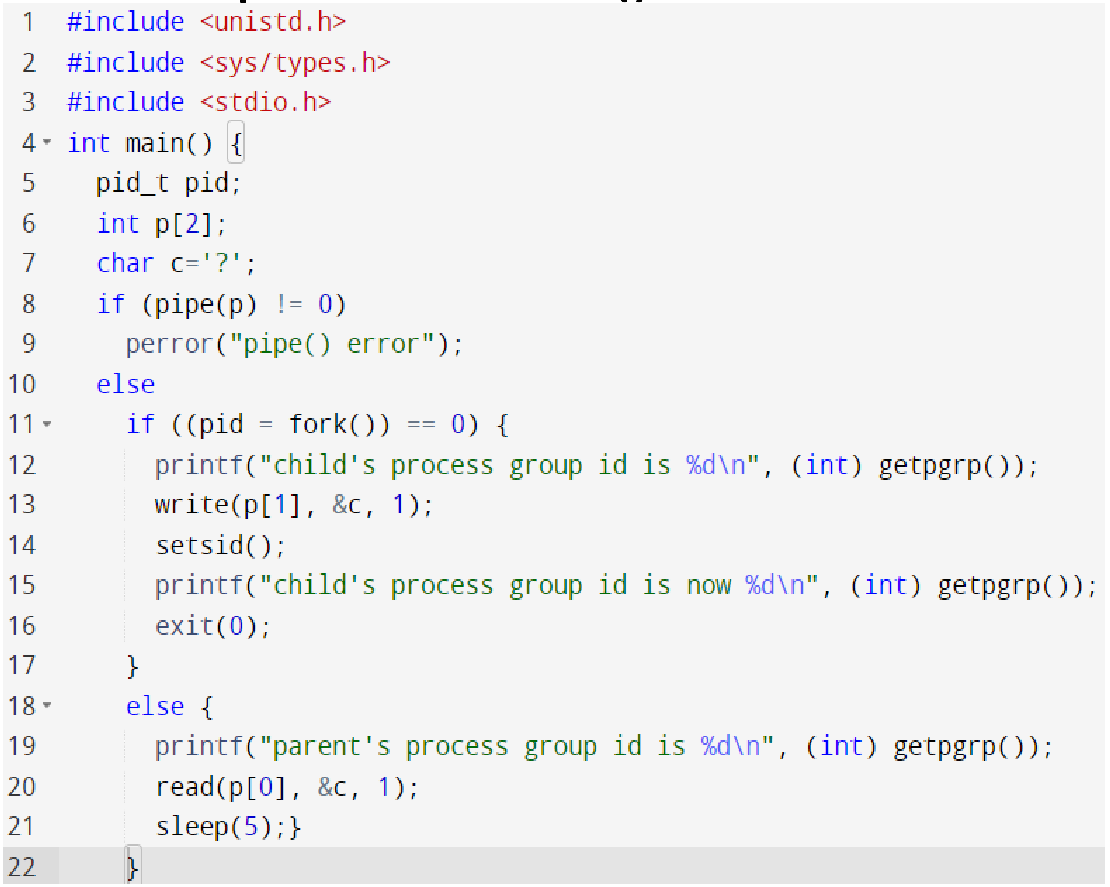
  The process groupid(pgid) of the new process group is equal to the process id(pid) of the caller. The caller starts as the only process in the new process group and in the new session.

setsid()

- **Returned value**
    - If successful, setsid() returns the value of the caller's new pgid.
    - If unsuccessful, setsid() returns -1 and sets errno to indicate the error.

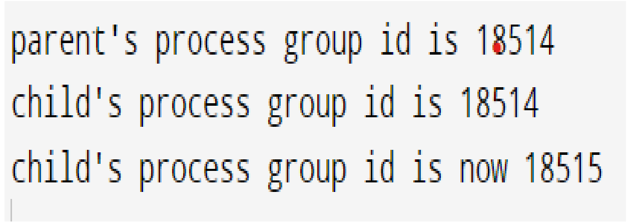
- **Occurrence of Errors**
    - The caller is already a process group leader.
    - The caller's pid matches the pgid of some other process.

Example of setsid()



```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main() {
  pid_t pid;
  int p[2];
  char c='?';
  if (pipe(p) != 0)
    perror("pipe() error");
  else
    if ((pid = fork()) == 0) {
      printf("child's process group id is %d\n", (int) getpgrp());
      write(p[1], &c, 1);
      setsid();
      printf("child's process group id is now %d\n", (int) getpgrp());
      exit(0);
    }
    else {
      printf("parent's process group id is %d\n", (int) getpgrp());
      read(p[0], &c, 1);
      sleep(5);}
    }
```

OUTPUT



```
parent's process group id is 18514
child's process group id is 18514
child's process group id is now 18515
```

getpgrp()

- System call to get the process group ID
- It is always successful
- There are no documented errno values

- <u>Signature</u> :-

  pid.t   getpgrp(void)

  Parameter : No parameter
  Return Value : returns the process group ID of the calling process
  Error Values : No return value is preserved to indicate an error
  Header Files : #include <unistd.h>

getpgrp() - Example to get all process group ids

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

int main() {
        int status;
        int pid = fork();
        if (pid == 0) {
                int cid = fork();
                if (cid == 0) {
                  cout<<"grandchild's pid is : "<<getpid()<<" and process group id is : "<<getpgrp()<<endl;
                  exit(0);
                }
                cout<<"child's pid is : "<<getpid()<<" and process group id is : "<<getpgrp()<<endl;
                wait(&status);
                exit(0);
        }
        cout<<"parent's pid is : "<<getpid()<<" and process group id is : "<<getpgrp()<<endl;
        cout<<"the parent's parent's pid is : "<<getppid()<<endl;
        wait(&status);
}
```

getpgrp() - Example to get all process group ids

- **Output** :-

```
parent's pid is : 68 and process group id is : 68
the parent's parent's pid is : 67
child's pid is : 72 and process group id is : 68
grandchild's pid is : 73 and process group id is : 68
```

getpgrp()

- It is equivalent to :-

  getpgid(pid = 0)

  If pid = 0 in getpgid(), it returns the process group ID of the calling process.

getpgrp() = getpgid(0)

- **Example** :-

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
#include <stdlib.h>
using namespace std;

int main() {
        int status;
        int pid = fork();
        if (pid == 0) {
                        cout<<"\nChild process pid : "<<getpid()<<endl;
                        cout<<"Process group id using getpgrp() : "<<getpgrp()<<endl;
                        cout<<"Process group id using getpgid(0) : "<<getpgid(0)<<endl;
        }
        else{
                        wait(NULL);
                        cout<<"\n\nParent process pid : "<<getpid()<<endl;
                        cout<<"Process group id using getpgrp() : "<<getpgrp()<<endl;
                        cout<<"Process group id using getpgid(0) : "<<getpgid(0)<<endl;
        }
        return 0;
}
```

Output :-

```
Child process pid : 2456
Process group id using getpgrp() : 2452
Process group id using getpgid(0) : 2452


Parent process pid : 2452
Process group id using getpgrp() : 2452
Process group id using getpgid(0) : 2452
```

getpgid()

- **Syntax**

  pid_t getpgid(pid_t *pid*)

- **Description**

  The getpgid function returns the process group id of the process with process id equal to *pid* .

  Value of 0 for the *pid* argument (*getpgid(0)*) return process group id of the calling process.
  Which is equivalent to *getpgrp()* function.

getpgid()

- **<u>Returned value</u>**
    - If successful, returns a process group ID of process *pid*.
    - If unsuccessful, It returns -1 and sets errno to indicate the error.

- **<u>Occurrence of Errors</u>**
    - One of the following error conditions exists:
        - There is no process with a process ID equal to *pid*.
        - The process *pid* is not in the same session as the calling process, and can't access the process group ID of that process from the calling process.

Example of getpgid()

```
pgid = getpgid(pid);   //pgid stores process group id of pid
        if ( pgid  == -1 ) {
        //error ocurred
                …}

        else if ( pgid ==  getpgid(0) ) {
        // pid is of same process group as calling process
        …}

                else{
                //pid is of different process group to calling process
        …}
```

setpgrp()

- Signature :-

  pid.t   setpgrp(void)

  Parameter : No parameter
  Return Value : returns the new process group ID
  Header Files : #include <sys/types.h>
  #include <unistd.h>

setpgrp()

- If the calling process is not already a session leader, setpgrp() sets the process group ID of the calling process to the process ID of the calling process.
- If setpgrp() creates a new session, then the new session has no controlling terminal.
- The setpgrp() function has no effect when the calling process is a session leader.
- Upon completion, *setpgrp()* returns the process group ID.
- No errors are defined.

setpgrp() - Example

- **Example** :-

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
using namespace std;

int main() {
        int status;
        int pid = fork();
        if (pid == 0) {
           cout<<"\nChild process pid : "<<getpid()<<endl;
           cout<<"Process group id using getpgrp() : "<<getpgrp()<<endl;
           cout<<"Process group id after setpgrp() : "<<setpgrp()<<endl;
           cout<<"Process group id using getpgrp() after setpgrp() : "<<getpgrp()<<endl;
        }
        else{
           wait(NULL);
           cout<<"\nParent process pid : "<<getpid()<<endl;
           cout<<"Process group id using getpgrp() : "<<getpgrp()<<endl;
        }
}
```

- **Output** :-

```
Child process pid : 1253
Process group id using getpgrp() : 1249
Process group id after setpgrp() : 1253
Process group id using getpgrp() after setpgrp() : 1253

Parent process pid : 1249
Process group id using getpgrp() : 1249
```

setpgrp()

- It is equivalent to :-

  setpgid(pid = 0, pgid = 0)

  If pid = 0 in getpgid(), the process ID of the calling process is used i.e. it means we have to set the pgid of the calling process.
  If pgid = 0 in getpgid(), the process group id of the calling process is used i.e. the process specified by pid becomes a process group leader.

setpgrp() = setpgid(0,0)

- **Example** :-

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
using namespace std;

int main() {
                int status;
                int pid = fork();
                if (pid == 0) {
                  cout<<"\nChild process pid : "<<getpid()<<endl;
                  cout<<"Process group id using getpgrp() : "<<
                getpgrp()<<endl;
                  cout<<"Process group id after setpgrp() : "
                <<setpgrp()<<endl;
                  cout<<"Process group id using getpgrp()
                after setpgrp() : "<<getpgrp()<<endl;
                }else{
                  wait(NULL);
                }
}
```

Output :-

```
Child process pid : 92
Process group id using getpgrp() : 88
Process group id after setpgrp() : 92
Process group id using getpgrp()
after setpgrp() : 92
```

- **Example** :-

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
using namespace std;

int main() {
                int status;
                int pid = fork();
                if (pid == 0) {
                  cout<<"\nChild process pid : "<<getpid()<<endl;
          cout<<"Process group id using getpgrp() :
        "<<getpgrp()<<endl;
          cout<<"Process group id after setpgid(0,0) :
        "<<setpgid(0,0)<<endl;
          cout<<"Process group id using getpgrp() after setpgid(0,0) :
        "<<getpgrp()<<endl;
                }
        else{
                  wait(NULL);
                }
}
```

Output :-

```
Child process pid : 94
Process group id using getpgrp() : 90
Process group id after setpgid(0,0) : 0
Process group id using getpgrp() after
setpgid(0,0) : 94
```

setpgid()

- **Syntax**

  int setpgid(pid_t pid,pid_t pgid)

- **Description**

  It is used either to join an existing process group or create a new process group within the session of the calling process.

  pid_t pid is the process id(pid) of the process whose pgid you want to change. This must either be the caller of setpgid() or one of its children, and it must be in the caller's session. It cannot be the pid of a session leader. If pid is zero, the system uses the pid of the process calling setpgid().

  setpgrp() is equivalent to setpgid(0,0).

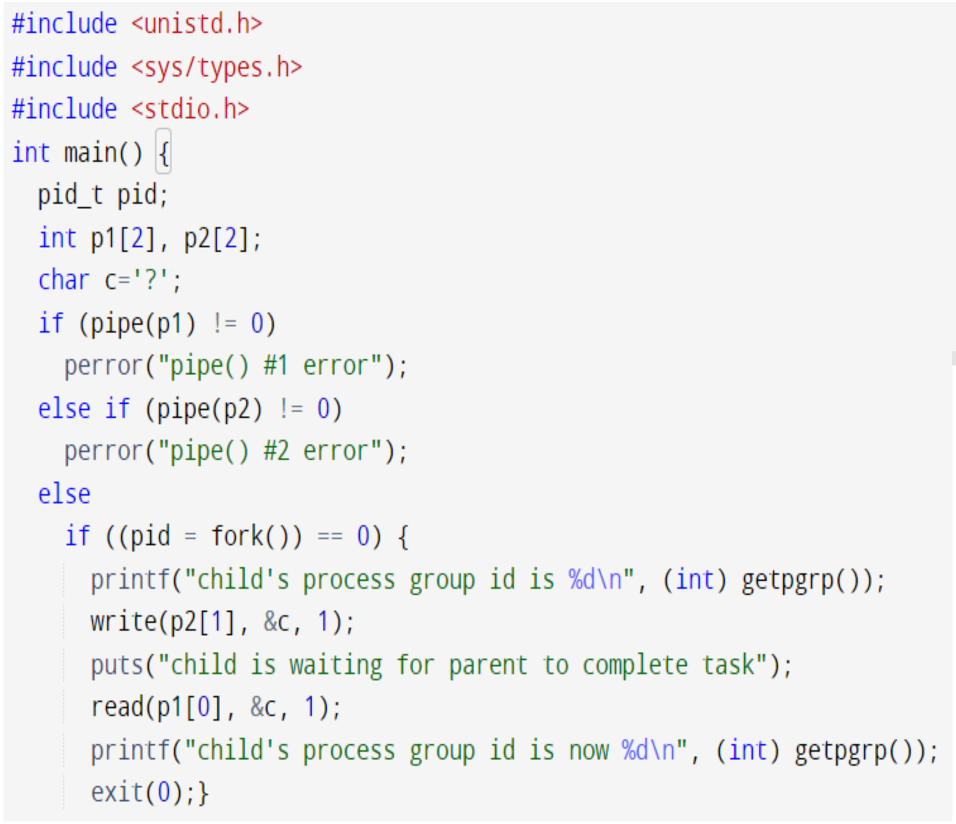
setpgid()

pid_t pgid is the new pgid you want to assign to the process identified by pid. If pgid indicates an existing process group, it must be in the caller's session. If pgid is zero, the system uses the pid of the process indicated by pid as the id for the new process group. The new group is created in the caller's session.

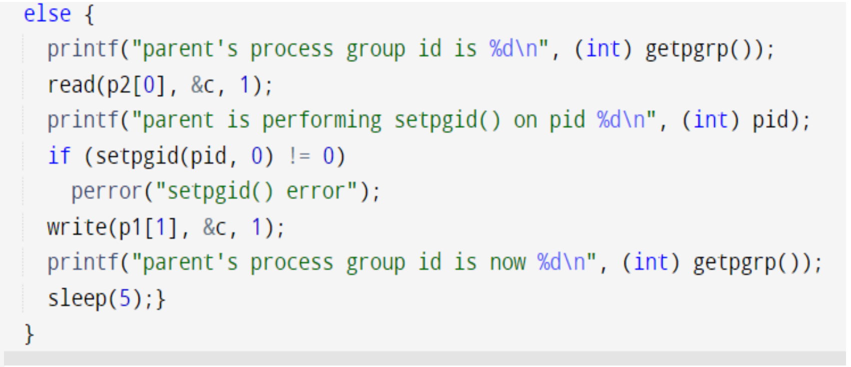
- **Returned value**
  - If successful, setpgid() returns 0.
  - If unsuccessful, setpgid() returns -1 and sets errno to indicate the error.

- **Occurrence of Errors**
  - pgid is less than zero or has some other unsupported value.
  - pid does not match the pid of the calling process or any of its children.
  - The caller cannot change the pgid of the specified process

Example of setpgid()

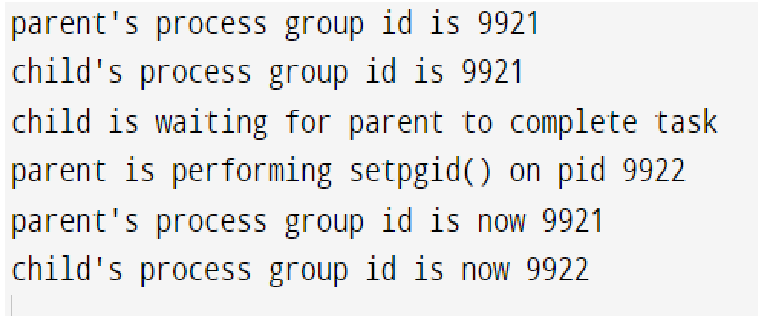


```c
#include <unistd.h>
#include <sys/types.h>
#include <stdio.h>
int main() {
  pid_t pid;
  int p1[2], p2[2];
  char c='?';
  if (pipe(p1) != 0)
    perror("pipe() #1 error");
  else if (pipe(p2) != 0)
    perror("pipe() #2 error");
  else
    if ((pid = fork()) == 0) {
      printf("child's process group id is %d\n", (int) getpgrp());
      write(p2[1], &c, 1);
      puts("child is waiting for parent to complete task");
      read(p1[0], &c, 1);
      printf("child's process group id is now %d\n", (int) getpgrp());
      exit(0);}
```



```c
else {
  printf("parent's process group id is %d\n", (int) getpgrp());
  read(p2[0], &c, 1);
  printf("parent is performing setpgid() on pid %d\n", (int) pid);
  if (setpgid(pid, 0) != 0)
    perror("setpgid() error");
  write(p1[1], &c, 1);
  printf("parent's process group id is now %d\n", (int) getpgrp());
  sleep(5);}
}
```

OUTPUT



```
parent's process group id is 9921
child's process group id is 9921
child is waiting for parent to complete task
parent is performing setpgid() on pid 9922
parent's process group id is now 9921
child's process group id is now 9922
```

kill()

- Signature :-

  pid.t   kill(pid.t   pid, int   sig)

  Parameter : 2 parameters

  1. pid = process id
  2. sig = signal number of signal to be sent

  Return Value : 0 if successful, -1 in case of error and sets errno

  Error Values : return value is preserved to indicate an error

  Header Files :   #include <sys/types.h>

  #include <signal.h>

kill()

- The kill() function **sends a signal** to a process or a group of processes specified by the pid.
- The **signal to be sent is specified by sig** and is either one from the list given in <signal.h> or 0.
- If **sig is 0** (the null signal), error checking is performed but no signal is actually sent. This can be used to check the validity of pid.
- If **pid is greater than 0**, sig will be sent to the process whose process ID is equal to pid.
- If **pid is 0**, sig will be sent to all processes whose process group ID is equal to the process group ID of the sender, and for which the process has permission to send a signal.
- If **pid is -1**, sig will be sent to all processes for which the process has permission to send a signal.
- If **pid is negative but not −1**, sig will be sent to all processes whose process group ID is equal to the absolute value of pid and for which the process has permission to send a signal.

kill()

- Upon **successful completion, 0 is returned**.

- Otherwise, **−1 is returned**, no signal is sent, and errno is set to indicate the error.

- **Errors** : The kill() function will fail if:
  1. **EINVAL** - The sig argument is not a valid signal number.
  2. **EPERM** - The sig argument is SIGKILL and the pid argument is -1 (that is, the calling process does not have permission to send the signal to any of the processes specified by pid).
  3. **ESRCH** - No process or process group can be found corresponding to that specified by pid.

kill() - Example (sig=0)

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
#include <signal.h>
using namespace std;
int main() {
                int status;
                int pid = fork();
                cout<<"pid : "<<pid<<endl;
                if (pid == 0) {
                  cout<<"\nChild process pid : "<<getpid()<<endl;
                }else{
                  cout<<"Parent process pid : "<<getpid()<<endl;
                  cout<<"Executing kill(pid, 0) : "<<kill(pid, 0)<<endl;
                  cout<<"After kill"<<endl;
                  wait(&status);
                  if (WIFSIGNALED(status))
                                if (WTERMSIG(status) == SIGTERM)
                                                cout<<"child was ended with a SIGTERM"<<endl;
                                else
                                                cout<<"child was ended with a
"<<WTERMSIG(status)<<" signal"<<endl;
                  else
                                cout<<"child was not ended with a signal"<<endl;
                }
}
```

Output :-

```
pid : 22
Parent process pid : 18
Executing kill(pid, 0) : 0
After kill
pid : 0

Child process pid : 22
child was not ended with a signal
```

Note : WIFSIGNALED() method is used to check whether a process exited due to any signal i.e. whether the child process exited abnormally. It takes in process status code as returned by wait() or waitpid() and returns True if the process exited due to a signal, otherwise returns False.

WTERMSIG() method is used to get the signal number which caused the process to exit.

kill() - Example (pid>0)

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
#include <signal.h>
using namespace std;

int main() {
                int status;
        int pid = fork();
                cout<<"pid : "<<pid<<endl;
                if (pid == 0) {
                  cout<<"\nChild process pid : "<<getpid()<<endl;
                }else{
                  cout<<"Parent process pid : "<<getpid()<<endl;
                  cout<<"Executing kill(pid, sig) : "<<kill(pid, SIGTERM)<<endl;
                  cout<<"After kill"<<endl;
                  wait(&status);
                  if (WIFSIGNALED(status))
              if (WTERMSIG(status) == SIGTERM)
                cout<<"child was ended with a SIGTERM"<<endl;
              else
                cout<<"child was ended with a "<<WTERMSIG(status)<<" signal"<<endl;
           else
              cout<<"child was not ended with a signal"<<endl;
                }
}
```

<u>Output</u> :-

```
pid : 81
Parent process pid : 77
Executing kill(pid, sig) : 0
After kill
child was ended with a SIGTERM
```

Note : If instead of SIGTERM, **SIGKILL** signal was used, output is :-

```
pid : 82
Parent process pid : 78
Executing kill(pid, sig) : 0
After kill
child was ended with a 9 signal
```

kill() - Example (pid=0)

```
#include <unistd.h>
#include <sys/wait.h>
#include <iostream>
#include <signal.h>
using namespace std;

int main() {
        int status;
        int pid = fork();
        cout<<"pid : "<<pid<<endl;
        if(pid>0){
           cout<<"Process group id of child : "<<getpgid(pid)<<endl;
        }
        if (pid == 0) {
           cout<<"\nChild process pid : "<<getpid()<<endl;
        }
        else{
           cout<<"Parent process pid : "<<getpid()<<endl;
           cout<<"Parent process group id : "<<getpgrp()<<endl;
           cout<<"Executing kill(0, sig) : "<<kill(0, SIGTERM)<<endl;
           cout<<"After kill"<<endl;
           wait(&status);
        }
}
```

Output :-

```
pid : 1705
Process group id of child : 1701
Parent process pid : 1701
Parent process group id : 1701
```