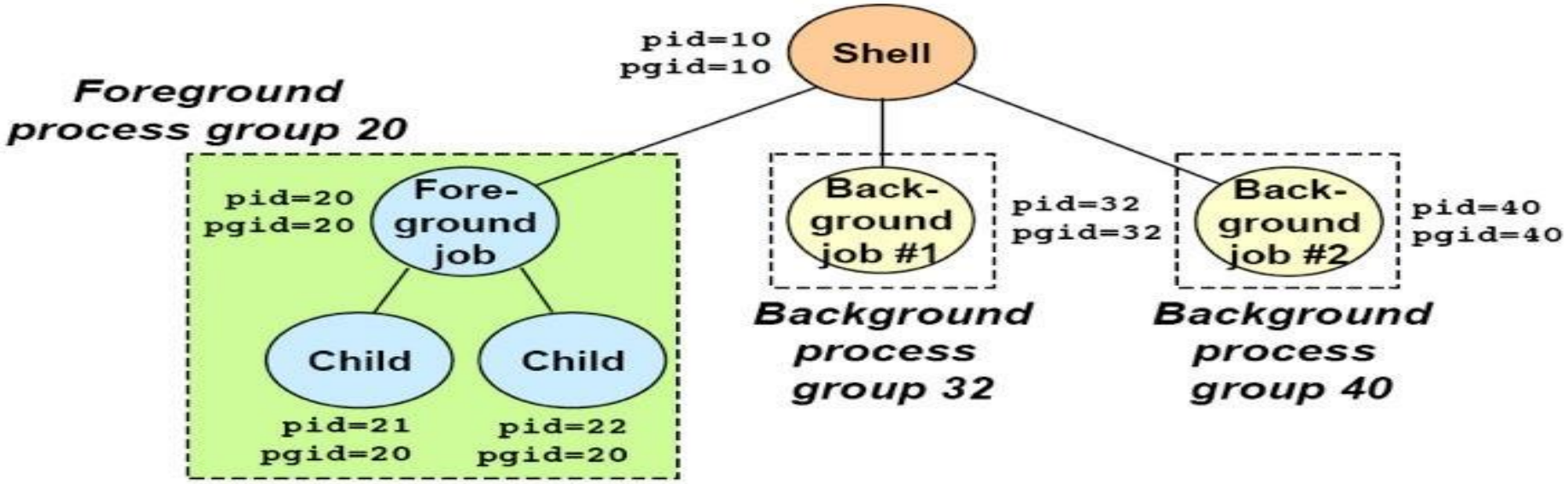
Process Groups and Control Terminals

- **Sending signals from the keyboard**
  - Typing **ctrl-c** (**ctrl-z**) sends a **SIGINT** (**SIGTSTP**) to every job in the foreground process group.
    - SIGINT: default action is to terminate each process.
    - SIGTSTP: default action is to stop (suspend) each process.



System Call setpgid() and getpgid()

System Call : pid_t setpgid(pid_t pid, pid_t pgrpId)

- setpgid() sets the process-group ID of the process with PID pid to pgrpId.
  - if pid is zero, the caller's process group ID is set to pgrpId.
  - In order for setpgid() to succeed and set the process group ID, at least one of the following conditions must be met:
    - The caller and the specified process must have the same owner.
    - The caller must be owned by a super-user.
- If setpgid() fails, it returns a value of -1.

System Call: pid_t getpgid(pid_t pid)

- getpgid()" returns the process group ID of the process with PID pid.
  - If pid is zero, the process group ID of the caller is returned.

System Call setpgid() and getpgid() Example

- example illustrates the fact that a terminal distributes signals to all of the processes in its control process' process group.
- Since the child inherited its process group from its parent, both the parent and child caught the SIGINT signal.

process_group_same.c

$ ./process_group_same.out parent pid 19354 and parent group 19354 waits

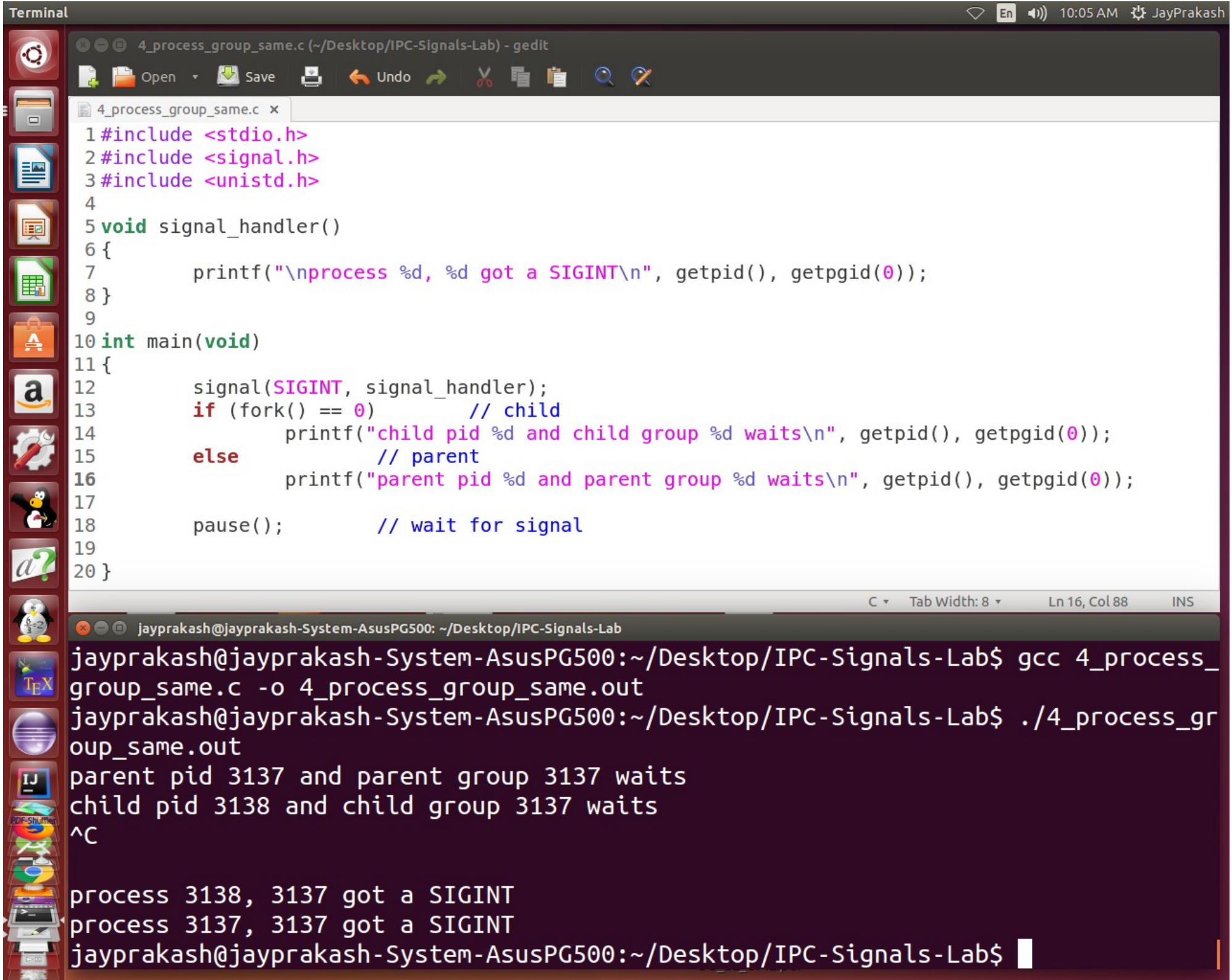
child pid 19355 and child group 19354 waits

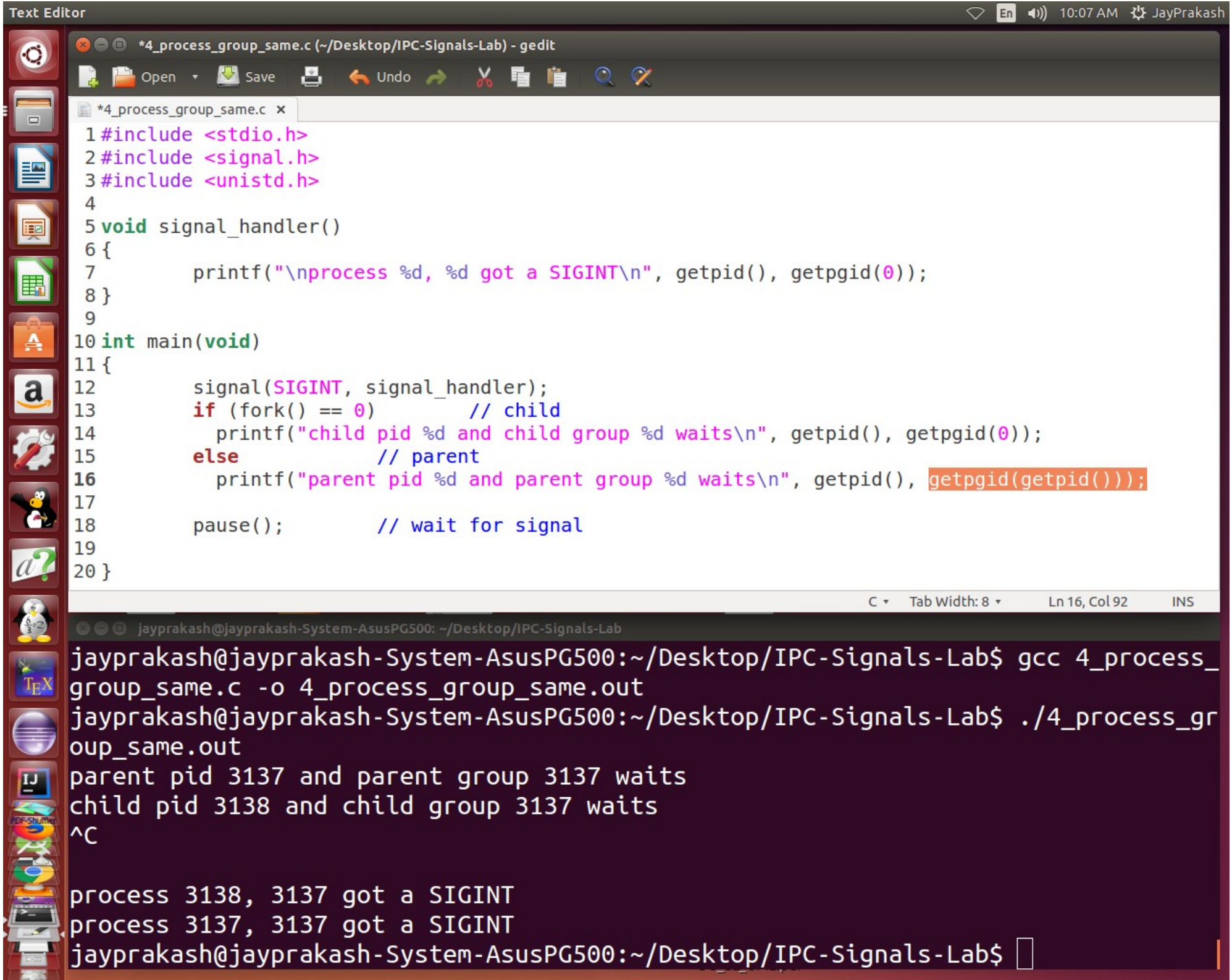
^C

process 19355 got a SIGINT

process 19354 got a SIGINT

$


Terminal
10:05 AM
JayPrakash
4_process_group_same.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
4_process_group_same.c
1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4
5 void signal_handler()
6 {
7 printf("\nprocess %d, %d got a SIGINT\n", getpid(), getpgid(0));
8 }
9
10 int main(void)
11 {
12 signal(SIGINT, signal_handler);
13 if (fork() == 0) // child
14 printf("child pid %d and child group %d waits\n", getpid(), getpgid(0));
15 else // parent
16 printf("parent pid %d and parent group %d waits\n", getpid(), getpgid(0));
17
18 pause(); // wait for signal
19
20 }
C
Tab Width: 8
Ln 16, Col 88
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ gcc 4_process_group_same.c -o 4_process_group_same.out
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./4_process_group_same.out
parent pid 3137 and parent group 3137 waits
child pid 3138 and child group 3137 waits
^C
process 3138, 3137 got a SIGINT
process 3137, 3137 got a SIGINT
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$

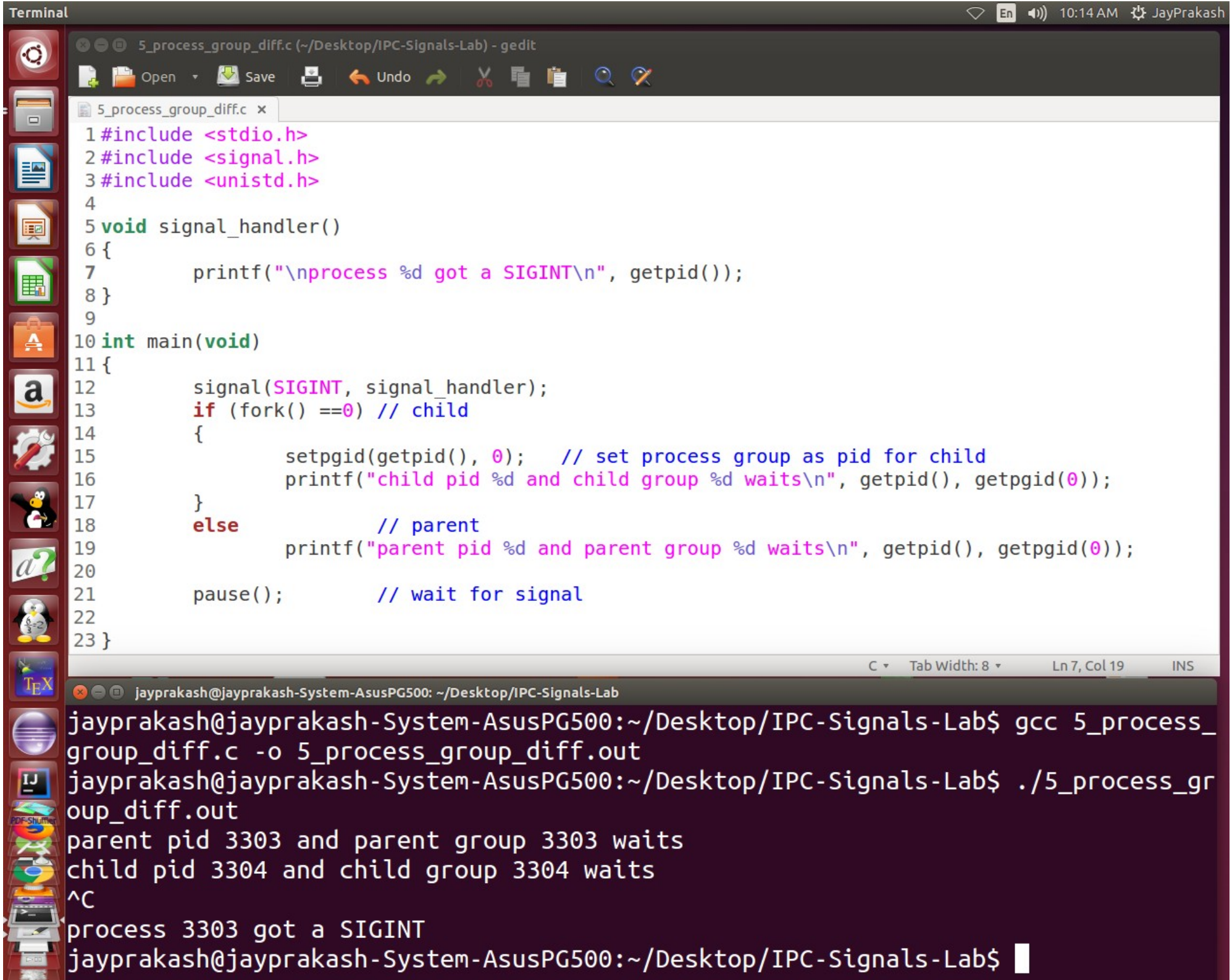

Text Editor
10:07 AM
JayPrakash
*4_process_group_same.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
*4_process_group_same.c
1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4
5 void signal_handler()
6 {
7 printf("\nprocess %d, %d got a SIGINT\n", getpid(), getpgid(0));
8 }
9
10 int main(void)
11 {
12 signal(SIGINT, signal_handler);
13 if (fork() == 0) // child
14 printf("child pid %d and child group %d waits\n", getpid(), getpgid(0));
15 else // parent
16 printf("parent pid %d and parent group %d waits\n", getpid(), getpgid(getpid()));
17
18 pause(); // wait for signal
19
20 }
C
Tab Width: 8
Ln 16, Col 92
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ gcc 4_process_group_same.c -o 4_process_group_same.out
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./4_process_group_same.out
parent pid 3137 and parent group 3137 waits
child pid 3138 and child group 3137 waits
^C
process 3138, 3137 got a SIGINT
process 3137, 3137 got a SIGINT
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$

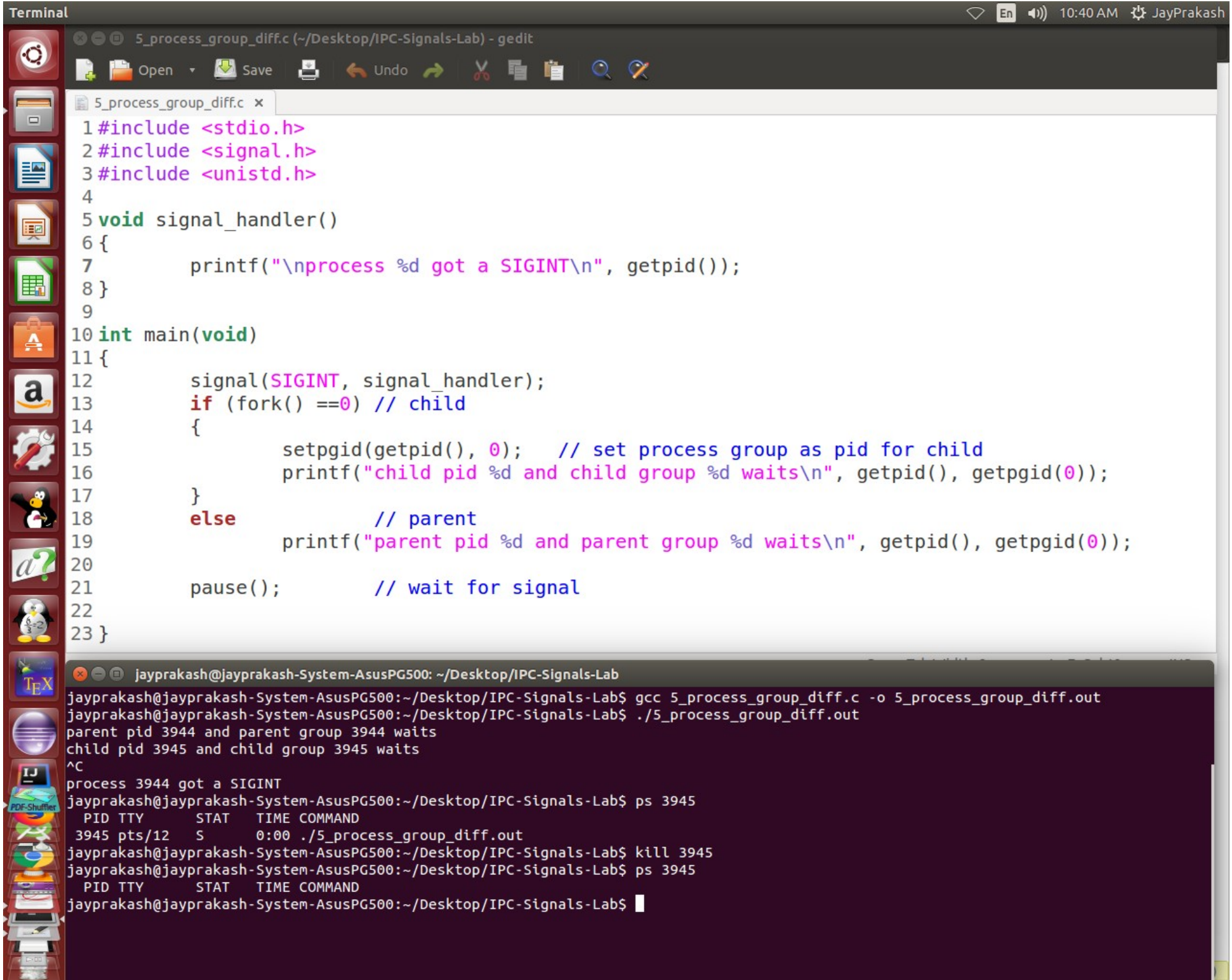
System Call setpgid() and getpgid() Example

- example illustrates the fact that a terminal distributes signals to all of the processes in its control process' process group.
- Since the child is in different process group SIGINT signal was caught only by parent and not child. We had to explicitly kill child since it was orphan.

Process_group_diff.c

```
$ ./process_group_diff.out
parent pid 19460 and parent group 19460 waits
child pid 19461 and child group 19461 waits
^C
process 19460 got a SIGINT
$ ps 19461
PID TTY      STAT   TIME COMMAND
19461 pts/1    S      0:00 ./process_group_diff.out
$ kill 19461
$ ps 19461
PID TTY      STAT   TIME COMMAND
$
```


Terminal
10:14 AM
JayPrakash
5_process_group_diff.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
5_process_group_diff.c
1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4
5 void signal_handler()
6 {
7 printf("\nprocess %d got a SIGINT\n", getpid());
8 }
9
10 int main(void)
11 {
12 signal(SIGINT, signal_handler);
13 if (fork() ==0) // child
14 {
15 setpgid(getpid(), 0); // set process group as pid for child
16 printf("child pid %d and child group %d waits\n", getpid(), getpgid(0));
17 }
18 else // parent
19 printf("parent pid %d and parent group %d waits\n", getpid(), getpgid(0));
20
21 pause(); // wait for signal
22
23 }
C
Tab Width: 8
Ln 7, Col 19
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ gcc 5_process_group_diff.c -o 5_process_group_diff.out
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./5_process_group_diff.out
parent pid 3303 and parent group 3303 waits
child pid 3304 and child group 3304 waits
^C
process 3303 got a SIGINT
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$


Terminal
10:40 AM
JayPrakash
5_process_group_diff.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
5_process_group_diff.c
1 #include <stdio.h>
2 #include <signal.h>
3 #include <unistd.h>
4
5 void signal_handler()
6 {
7 printf("\nprocess %d got a SIGINT\n", getpid());
8 }
9
10 int main(void)
11 {
12 signal(SIGINT, signal_handler);
13 if (fork() ==0) // child
14 {
15 setpgid(getpid(), 0); // set process group as pid for child
16 printf("child pid %d and child group %d waits\n", getpid(), getpgid(0));
17 }
18 else // parent
19 printf("parent pid %d and parent group %d waits\n", getpid(), getpgid(0));
20
21 pause(); // wait for signal
22
23 }
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ gcc 5_process_group_diff.c -o 5_process_group_diff.out
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./5_process_group_diff.out
parent pid 3944 and parent group 3944 waits
child pid 3945 and child group 3945 waits
^C
process 3944 got a SIGINT
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ps 3945
PID TTY STAT TIME COMMAND
3945 pts/12 S 0:00 ./5_process_group_diff.out
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ kill 3945
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ps 3945
PID TTY STAT TIME COMMAND
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$

Sending Signal from another Process: System Call kill()

- A process may send a signal to another process by using the kill() system call
- kill() is a misnomer, since many of the signals that it can send to do not terminate a process
- Its call kill() because it was originally designed to terminate a process which is still one of its function

System Call kill()

System Call: int kill(pid_t pid, int sigCode)

- kill() sends the signal with value sigCode to the process with PID pid.
- kill() succeeds and the signal is sent as long as at least one of the following conditions is satisfied:
  - The sending process and the receiving process have the same owner.
  - The sending process is owned by a super-user.
- There are a few variations on the way that “kill()” works:
  - If pid is positive, then *signal* is sent to the process with the ID specified by *pid*
  - If pid is zero, then *signal* is sent to every process in the same process group as of the calling process.
  - If pid is -1 and the sender is owned by a super-user, the signal is sent to all processes, including the sender.
  - If pid is -1 and the sender is not owned by a super-user, the signal is sent to all of the processes owned by the same owner as that of the sender, excluding the sending process.

Detecting death of a Child Process

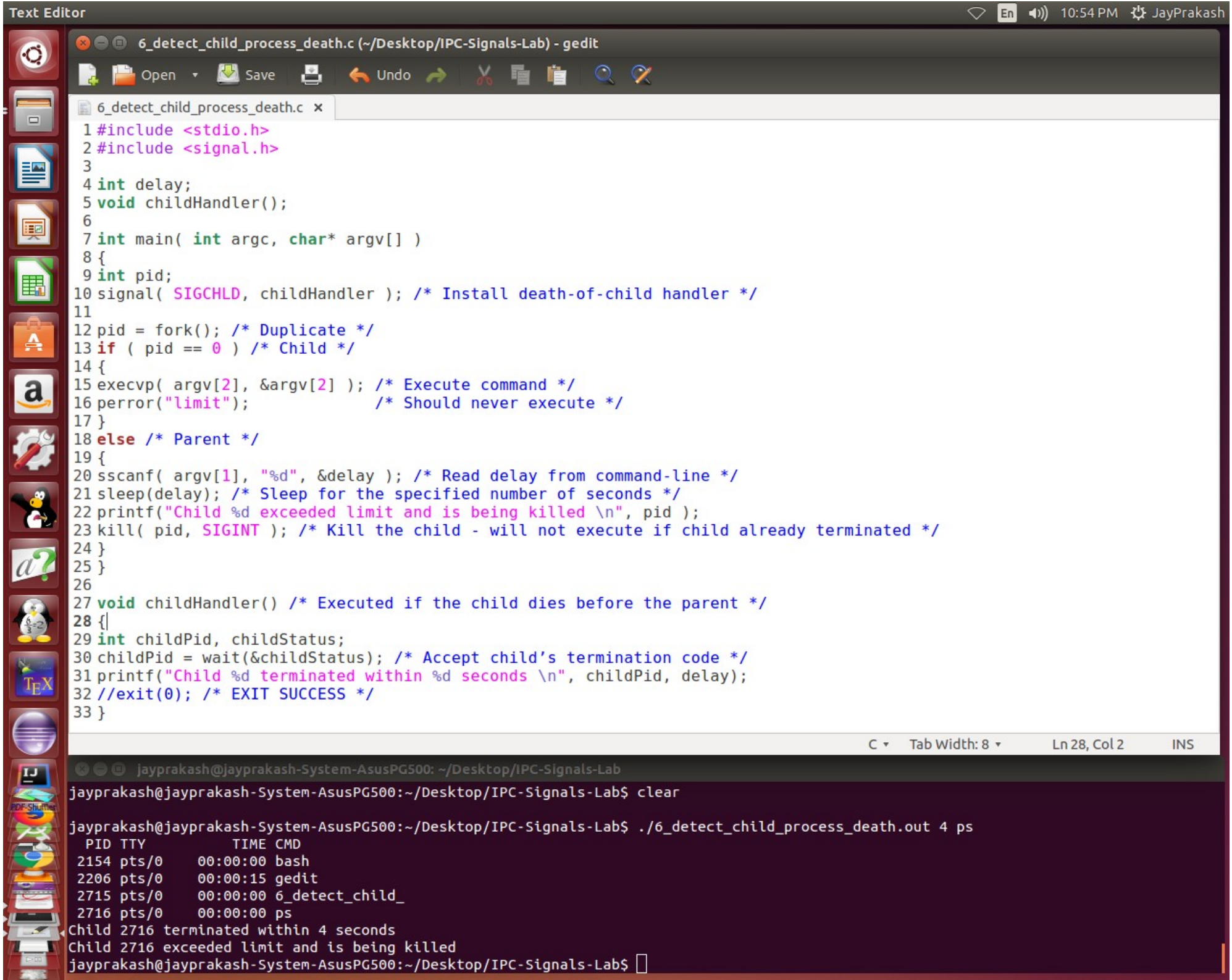
- When a parent's child terminates, the child process sends its parent a SIGCHLD signal.
- A parent process often installs a handler to deal with this signal; this handler typically executes a wait() system call to accept the child's termination code and let the child dezombify.
- the parent can choose to ignore SIGCHLD signals, in which case the child dezombifies automatically.
- The program works by performing the following steps:
  - The parent process installs a SIGCHLD handler that is executed when its child process terminates.
  - The parent process forks a child process to execute the command.
  - The parent process sleeps for the specified number of seconds. when it wakes up, it sends its child process a SIGINT signal to kill it.
  - If the child terminates before its parent finishes sleeping, the parent's SIGCHLD handler is executed, causing the parent to terminate immediately.

Detecting death of a Child Process

```
#include <stdio.h>
#include <signal.h>
int delay;
void childHandler();
int main( int argc, char* argv[] )
{
   int pid;
   signal( SIGCHLD, childHandler );  /* Install death-of-child handler */
   pid = fork();  /* Duplicate */
   if ( pid == 0 )  /* Child */
   {
        execvp( argv[2], &argv[2] );   /* Execute command */
        perror("limit");   /* Should never execute */
   }
else   /* Parent */
 {
  sscanf( argv[1], "%d", &delay );  /* Read delay from command-line */
  sleep(delay);   /* Sleep for the specified number of seconds */
  printf("Child %d exceeded limit and is being killed \n", pid );
  kill( pid, SIGINT );   /* Kill the child */
 }
}
```

Detecting death of a Child Process

```
void childHandler()  /* Executed if the child dies before the parent */
{
   int  childPid, childStatus;
   childPid = wait(&childStatus);   /* Accept child's termination code */
   printf("Child %d terminated within %d seconds \n", childPid, delay);
   exit(/* EXIT SUCCESS */  0);
}
```

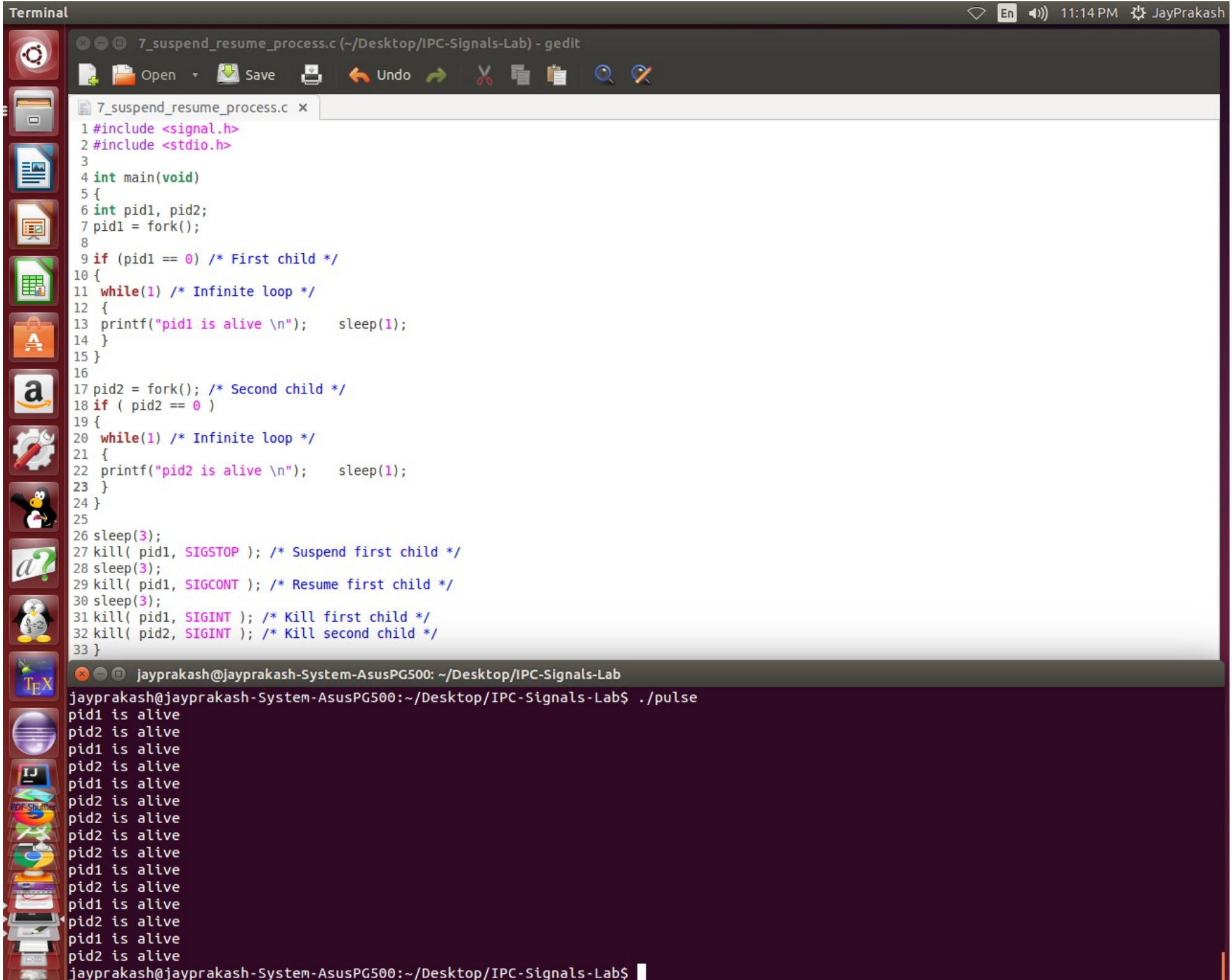

Text Editor
10:54 PM
JayPrakash
6_detect_child_process_death.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
6_detect_child_process_death.c
1 #include <stdio.h>
2 #include <signal.h>
3
4 int delay;
5 void childHandler();
6
7 int main(int argc, char* argv[])
8 {
9 int pid;
10 signal(SIGCHLD, childHandler); /* Install death-of-child handler */
11
12 pid = fork(); /* Duplicate */
13 if (pid == 0) /* Child */
14 {
15 execvp(argv[2], &argv[2]); /* Execute command */
16 perror("limit"); /* Should never execute */
17 }
18 else /* Parent */
19 {
20 sscanf(argv[1], "%d", &delay); /* Read delay from command-line */
21 sleep(delay); /* Sleep for the specified number of seconds */
22 printf("Child %d exceeded limit and is being killed \n", pid);
23 kill(pid, SIGINT); /* Kill the child - will not execute if child already terminated */
24 }
25 }
26
27 void childHandler() /* Executed if the child dies before the parent */
28 {
29 int childPid, childStatus;
30 childPid = wait(&childStatus); /* Accept child's termination code */
31 printf("Child %d terminated within %d seconds \n", childPid, delay);
32 //exit(0); /* EXIT SUCCESS */
33 }
C
Tab Width: 8
Ln 28, Col 2
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ clear
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./6_detect_child_process_death.out 4 ps
PID TTY TIME CMD
2154 pts/0 00:00:00 bash
2206 pts/0 00:00:15 gedit
2715 pts/0 00:00:00 6_detect_child_
2716 pts/0 00:00:00 ps
Child 2716 terminated within 4 seconds
Child 2716 exceeded limit and is being killed
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$

Suspending and Resuming Process using SIGSTOP and SIGCONT signals

- The SIGSTOP and SIGCONT signals suspend and resume a process, respectively.
- They are used by the UNIX shells that support job control to implement built-in commands like stop, fg, and bg.
- Program in next slide performs these steps:
  - The main program creates two children; they both enter an infinite loop and display a message every second.
  - The main program waits for three seconds and then suspend the first child.
  - After another three seconds, the parent restarts the first child, waits a little while longer, and then terminated both children.

Suspending and Resuming Process using SIGSTOP and SIGCONT signals

```
#include <signal.h>
#include <stdio.h>
int main(void)
{ int pid1;
  int pid2;
  pid1 = fork();
  if (pid1== 0) /* First child */
   {
     while(1)  /* Infinite loop */
      {
        printf("pid1 is alive \n");
        sleep(1);
      }
   }
  pid2 = fork();  /* Second child */
if ( pid2 == 0 )
   {
     while(1)   /* Infinite loop */
      {
        printf("pid2 is alive \n");
        sleep(1);
      }
   }
 sleep(3);
 kill( pid1, SIGSTOP );   /* Suspend first child */
 sleep(3);
 kill( pid1, SIGCONT );  /* Resume first child */
 sleep(3);
 kill( pid1, SIGINT );     /* Kill first child */
 kill( pid2, SIGINT );     /* Kill second child */
}
```

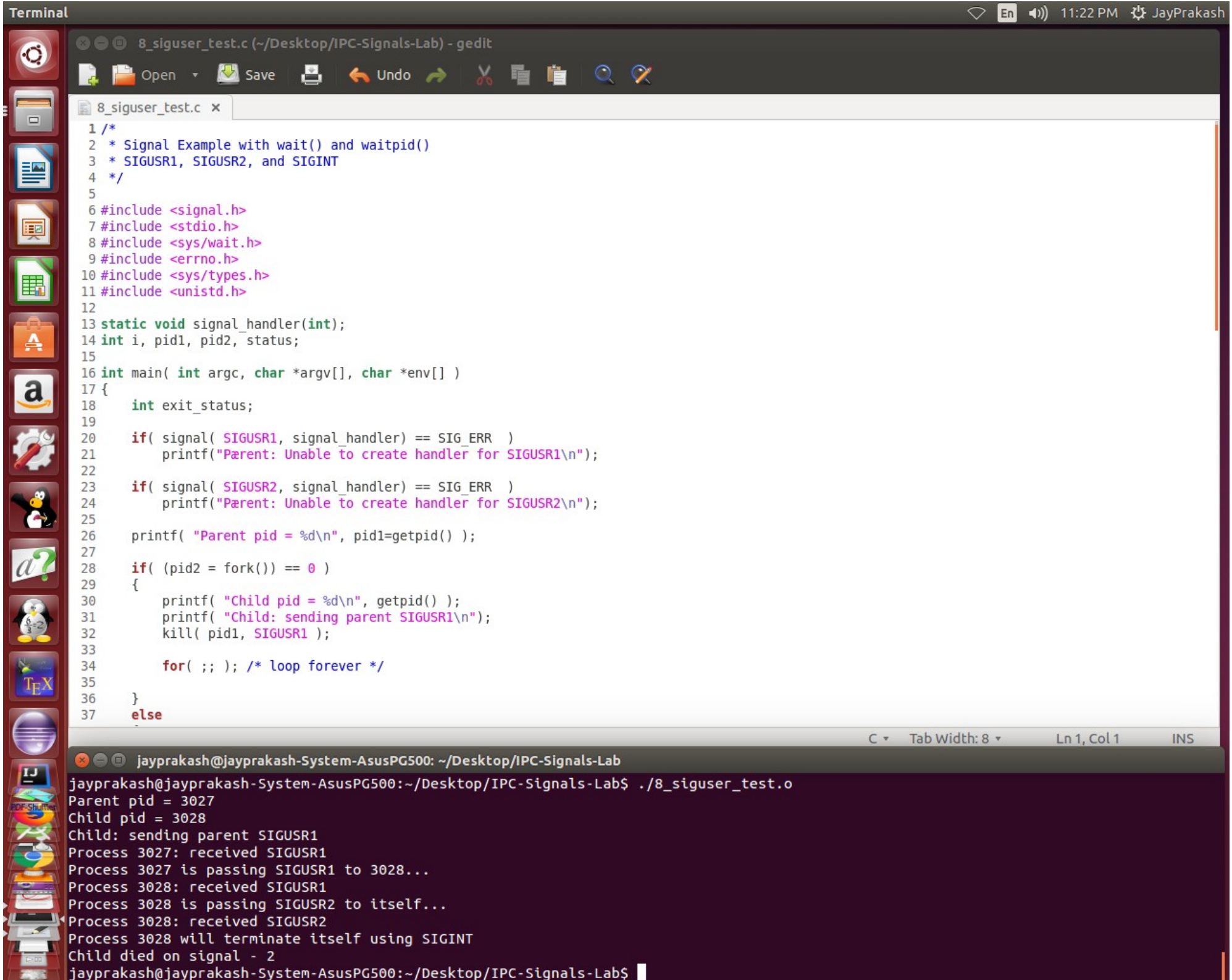

Terminal
En
11:14 PM
JayPrakash
7_suspend_resume_process.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
7_suspend_resume_process.c
1 #include <signal.h>
2 #include <stdio.h>
3
4 int main(void)
5 {
6 int pid1, pid2;
7 pid1 = fork();
8
9 if (pid1 == 0) /* First child */
10 {
11 while(1) /* Infinite loop */
12 {
13 printf("pid1 is alive \n"); sleep(1);
14 }
15 }
16
17 pid2 = fork(); /* Second child */
18 if (pid2 == 0)
19 {
20 while(1) /* Infinite loop */
21 {
22 printf("pid2 is alive \n"); sleep(1);
23 }
24 }
25
26 sleep(3);
27 kill(pid1, SIGSTOP); /* Suspend first child */
28 sleep(3);
29 kill(pid1, SIGCONT); /* Resume first child */
30 sleep(3);
31 kill(pid1, SIGINT); /* Kill first child */
32 kill(pid2, SIGINT); /* Kill second child */
33 }
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./pulse
pid1 is alive
pid2 is alive
pid1 is alive
pid2 is alive
pid1 is alive
pid2 is alive
pid2 is alive
pid2 is alive
pid2 is alive
pid1 is alive
pid2 is alive
pid1 is alive
pid2 is alive
pid1 is alive
pid2 is alive
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$

SIGUSR1 and SIGUSR2 → User Defined Signals

- Developer can use SIGUSR1 and/or SIGUSR2 to create own signals
- Steps for implementing SIGUSR1/2:
  - Process designated as recipient of SIGUSR1/2:
    - First create a signal handler function → void my_signal_handler(int signum)
      - signum will be passed as SIGUSR1 or SIGUSR2 depending on which signal was sent
    - Register signal handler function for each signal so that it can be called upon receipt of that signal → signal(SIGUSR1, my_signal_handler)
  - Process designated for sending signals SIGUSR1/2:
    - Uses kill() system call to generate/send signal to a specific process using pid → kill(pid1, SIGUSR1)
    - As we studied earlier kill() system call is misnomer and you can use it to send any signal to a process, in this case it is sending SIGUSR1 signal to pid1

Inter-Process Communication Using SIGUSR1 and SIGUSR1

- Inter Process Communication between parent and child using SIGUSR1 and SIGUSR2
- This example shows
- Register signal handler for SIGUSR1 and SIGUSR2 for parent process which inherited by child process because they belong to the same process group
- Child first sends SIGUSR1 to parent
- Parent upon receiving SIGUSR1 sends SIGUSR1 to child
- Child upon receiving SIGUSR1 sends SIGUSR2 to parent
- Parent upon receiving SIGUSR2 sends SIGUSR2 to child
- Child upon receiving SIGUSR2 terminates itself by sending SIGINT signal to itself
- Parent when detect child has died it termininates
- siguser_test.c


Terminal
11:22 PM
JayPrakash
8_siguser_test.c (~/Desktop/IPC-Signals-Lab) - gedit
Open
Save
Undo
8_siguser_test.c
1 /*
2 * Signal Example with wait() and waitpid()
3 * SIGUSR1, SIGUSR2, and SIGINT
4 */
5
6 #include <signal.h>
7 #include <stdio.h>
8 #include <sys/wait.h>
9 #include <errno.h>
10 #include <sys/types.h>
11 #include <unistd.h>
12
13 static void signal_handler(int);
14 int i, pid1, pid2, status;
15
16 int main(int argc, char *argv[], char *env[])
17 {
18 int exit_status;
19
20 if(signal(SIGUSR1, signal_handler) == SIG_ERR)
21 printf("Pærent: Unable to create handler for SIGUSR1\n");
22
23 if(signal(SIGUSR2, signal_handler) == SIG_ERR)
24 printf("Pærent: Unable to create handler for SIGUSR2\n");
25
26 printf("Parent pid = %d\n", pid1=getpid());
27
28 if((pid2 = fork()) == 0)
29 {
30 printf("Child pid = %d\n", getpid());
31 printf("Child: sending parent SIGUSR1\n");
32 kill(pid1, SIGUSR1);
33
34 for(;;); /* loop forever */
35
36 }
37 else
C
Tab Width: 8
Ln 1, Col 1
INS
jayprakash@jayprakash-System-AsusPG500: ~/Desktop/IPC-Signals-Lab
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$ ./8_siguser_test.o
Parent pid = 3027
Child pid = 3028
Child: sending parent SIGUSR1
Process 3027: received SIGUSR1
Process 3027 is passing SIGUSR1 to 3028...
Process 3028: received SIGUSR1
Process 3028 is passing SIGUSR2 to itself...
Process 3028: received SIGUSR2
Process 3028 will terminate itself using SIGINT
Child died on signal - 2
jayprakash@jayprakash-System-AsusPG500:~/Desktop/IPC-Signals-Lab$

Ignore Other Signals inside Signal Handler

```
int main(int ac, char *av[])
{
        void inthandler(int);
        void quithandler(int);
        char input[100];
        signal( SIGINT, inthandler ); // set trap
        signal( SIGQUIT, quithandler ); // set trap
        int i=1;
        while (i<5) {
                sleep(1);
                printf("main:%d\n",i++);
        }
}
```

```
void quithandler(int s) {
printf(" Received signal %d .. waiting\n", s );
.....
printf(" Leaving quithandler \n");
}
void inthandler(int s) {
signal(SIGQUIT, SIG_IGN);
printf(" Received signal %d .. waiting\n", s );
.....
printf(" Leaving inthandler \n");
signal( SIGQUIT, quithandler );
}
```