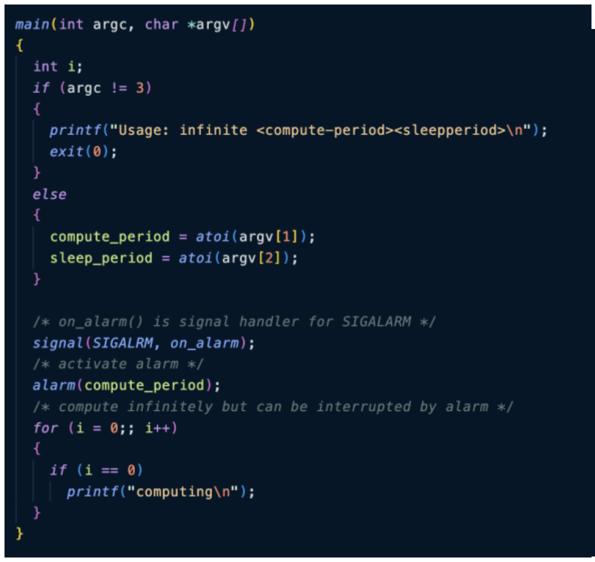
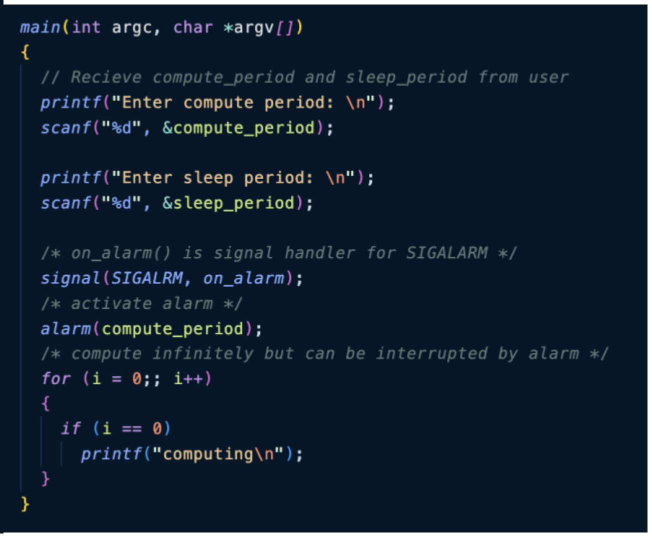
Activity 3

1.

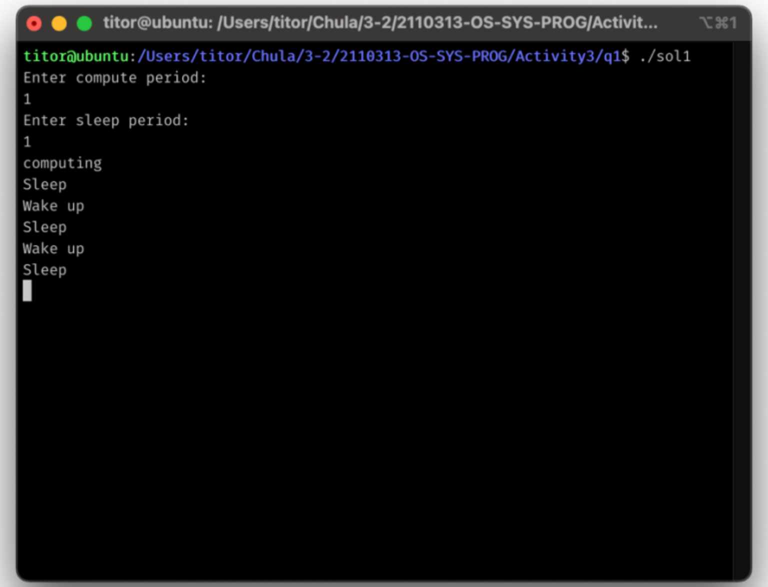
เปลี่ยนการรับ compute_period และ sleep_period จากการรับจาก argument เป็นรับจาก user input แทน



```c
main(int argc, char *argv[])
{
  int i;
  if (argc != 3)
  {
    printf("Usage: infinite <compute-period><sleepperiod>\n");
    exit(0);
  }
  else
  {
    compute_period = atoi(argv[1]);
    sleep_period = atoi(argv[2]);
  }

  /* on_alarm() is signal handler for SIGALARM */
  signal(SIGALRM, on_alarm);
  /* activate alarm */
  alarm(compute_period);
  /* compute infinitely but can be interrupted by alarm */
  for (i = 0;; i++)
  {
    if (i == 0)
      printf("computing\n");
  }
}
```



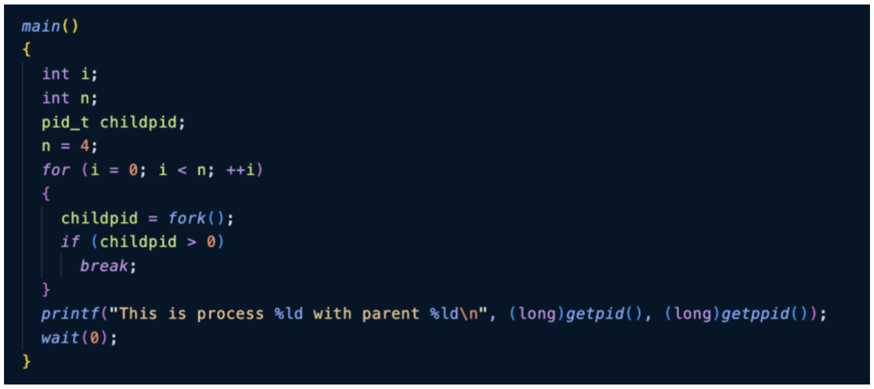
```c
main(int argc, char *argv[])
{
  // Recieve compute_period and sleep_period from user
  printf("Enter compute period: \n");
  scanf("%d", &compute_period);

  printf("Enter sleep period: \n");
  scanf("%d", &sleep_period);

  /* on_alarm() is signal handler for SIGALARM */
  signal(SIGALRM, on_alarm);
  /* activate alarm */
  alarm(compute_period);
  /* compute infinitely but can be interrupted by alarm */
  for (i = 0;; i++)
  {
    if (i == 0)
      printf("computing\n");
  }
}
```



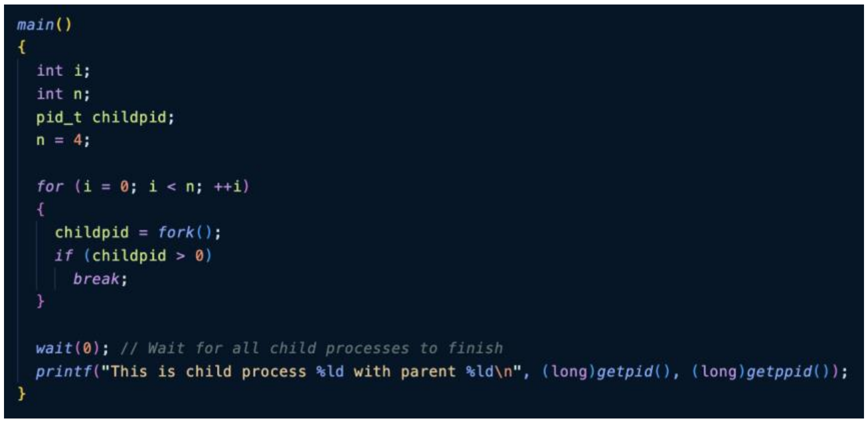
```
titor@ubuntu:/Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity3/q1$ ./sol1
Enter compute period:
1
Enter sleep period:
1
computing
Sleep
Wake up
Sleep
Wake up
Sleep
```

2.

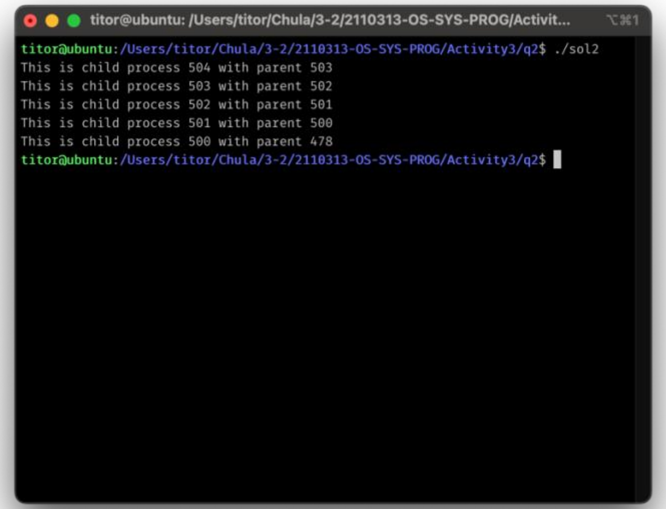
ทำการสลับระหว่าง wait(0) และ printf เพื่อรอให้ child process ทั้งหมดทำงานเสร็จก่อน เวลาแสดงผลจะได้แสดงเรียง จาก pid ที่มีค่ามากๆ แสดงผลออกมาก่อน



```c
main()
{
  int i;
  int n;
  pid_t childpid;
  n = 4;
  for (i = 0; i < n; ++i)
  {
    childpid = fork();
    if (childpid > 0)
      break;
  }
  printf("This is process %ld with parent %ld\n", (long)getpid(), (long)getppid());
  wait(0);
}
```



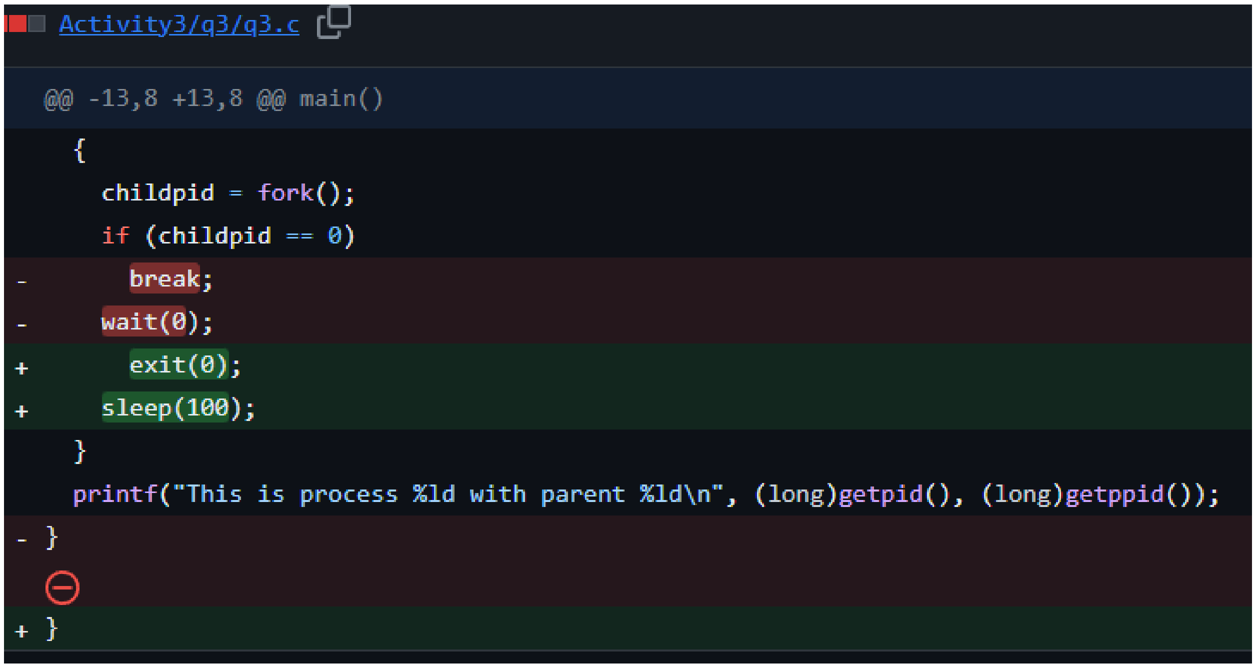
```c
main()
{
  int i;
  int n;
  pid_t childpid;
  n = 4;

  for (i = 0; i < n; ++i)
  {
    childpid = fork();
    if (childpid > 0)
      break;
  }

  wait(0); // Wait for all child processes to finish
  printf("This is child process %ld with parent %ld\n", (long)getpid(), (long)getppid());
}
```



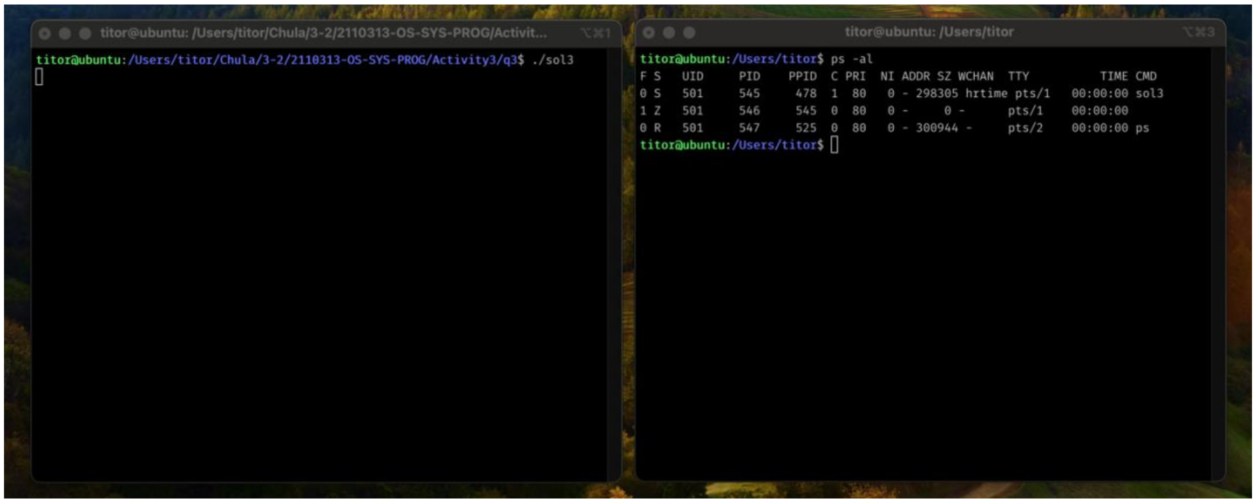
```
titor@ubuntu: /Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activit...   ⌥⌘1
titor@ubuntu:/Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity3/q2$ ./sol2
This is child process 504 with parent 503
This is child process 503 with parent 502
This is child process 502 with parent 501
This is child process 501 with parent 500
This is child process 500 with parent 478
titor@ubuntu:/Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity3/q2$ 
```

3.

Zombie process is fork process that child process died or exit before their parent does, to accomplish that by modify given source code. I simply add exit(0) instead of break when detect that it's child process, meanwhile I tell parent to sleep, to make sure that it won't exit before child does.



```
Activity3/q3/q3.c

@@ -13,8 +13,8 @@ main()
    {
      childpid = fork();
      if (childpid == 0)
-       break;
-     wait(0);
+       exit(0);
+     sleep(100);
    }
    printf("This is process %ld with parent %ld\n", (long)getpid(), (long)getppid());
- }

+ }
```



```
titor@ubuntu: /Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activit...

titor@ubuntu:/Users/titor/Chula/3-2/2110313-OS-SYS-PROG/Activity3/q3$ ./sol3
```

```
titor@ubuntu: /Users/titor

titor@ubuntu:/Users/titor$ ps -al
F S   UID     PID    PPID  C PRI  NI ADDR SZ WCHAN  TTY          TIME CMD
0 S   501     545     478  1  80   0 - 298305 hrtime pts/1   00:00:00 sol3
1 Z   501     546     545  0  80   0 -     0 -      pts/1    00:00:00
0 R   501     547     525  0  80   0 - 300944 -     pts/2    00:00:00 ps
titor@ubuntu:/Users/titor$
```