6. (20 points) Concurrent tasks and communication using pipes

a) Draw the tasks and the pipes between them as described by the following code segment.

b) Draw the file/pipe descriptor table for the parent and the child with the relevant points indicated.

c) What is the output?

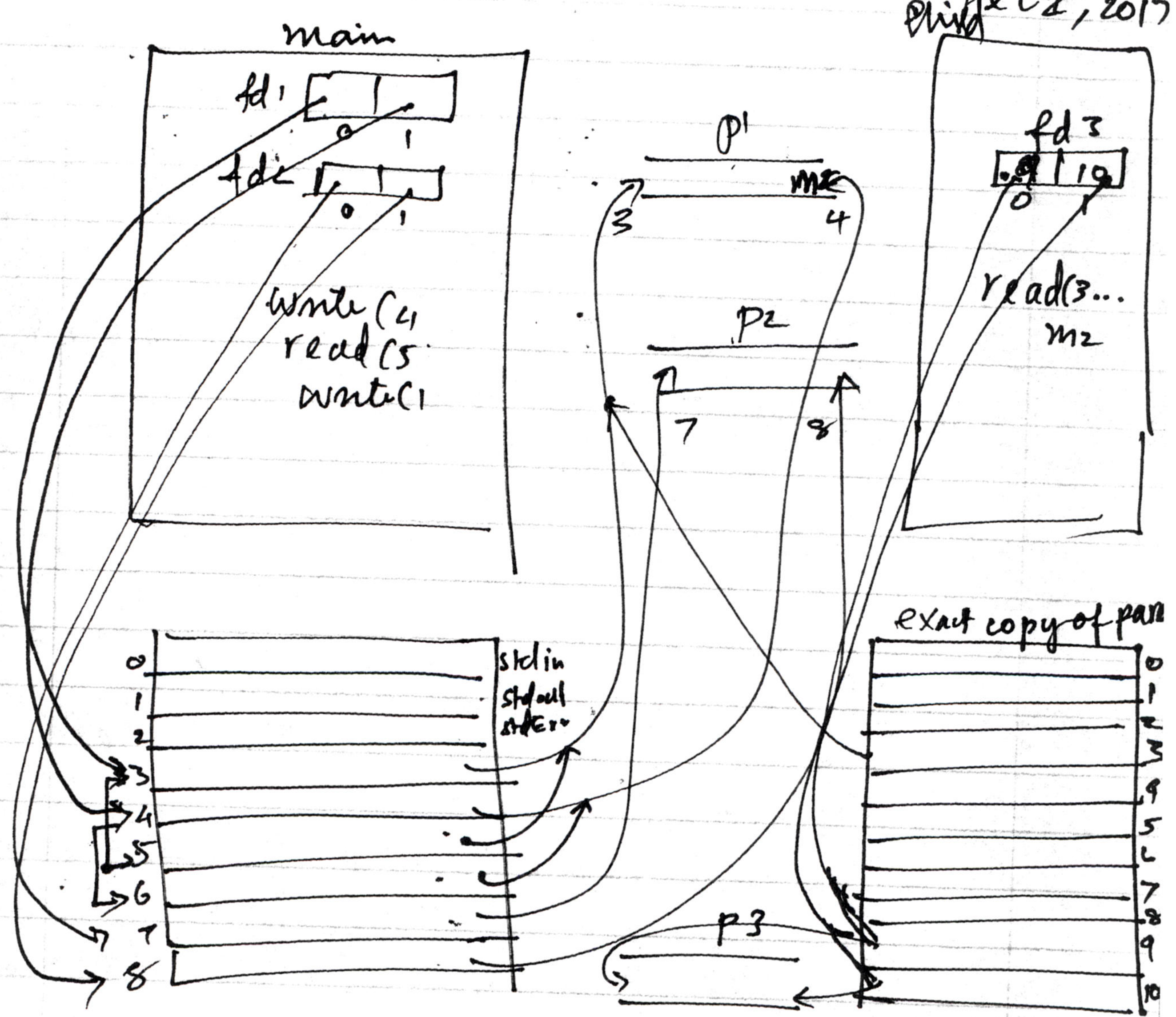
Also show the output, pipe structures and table structure at the point indicated: #1 ..... #4

Ch.8, Commx Synch.

p. 187-191 pipes

```
int main() {
pid_t newpid;
 int fd1[2], fd2[2];

 char m1[] = "Final Exam 2013\n";
  char m2[] = "Today is 10th December\n";

  char rbuf1[256];
  char rbuf2[256];
  int cn1, cn2;

  if ((pipe(fd1)== -1)) printf(" error \n");
  printf("fd1 %d %d  \n", fd1[0], fd1[1]);

  dup(fd1[0]);
  dup(fd1[1]);          //#1 show the open file descriptor table here      5

  write(4, m1, sizeof(m1));
  cn1 = read(5, rbuf1, 256);
  write(1, rbuf1, cn1);  //#2 show the structure of pipe fd1 with the respect the current process
                                                                              2
 if ((pipe(fd2)== -1)) printf(" error \n");
  printf("fd1 %d %d  \n", fd2[0], fd2[1]);

  if ((newpid = fork()) == -1) { printf("error \n"); return 0;}

 if (newpid >0 )
   {  //parent
   write(4, m2, sizeof(m2)); //#3 show the open file descriptor table here   write(8, "Done");
 }
 else { //child
   cn2 = read(3, rbuf2, 256);
   write(1, rbuf2,cn2);

   int fd3[2];
   if ((pipe(fd3)== -1)) printf(" error \n");
   printf("fd1 %d %d  \n", fd3[0], fd3[1]);    //#4 Show the open file descriptor table for parent + child
  // other code
 }                                       read (7, rubuf)
                                         write (1, . . .
  return 0; }}
```

Dec 1, 2017



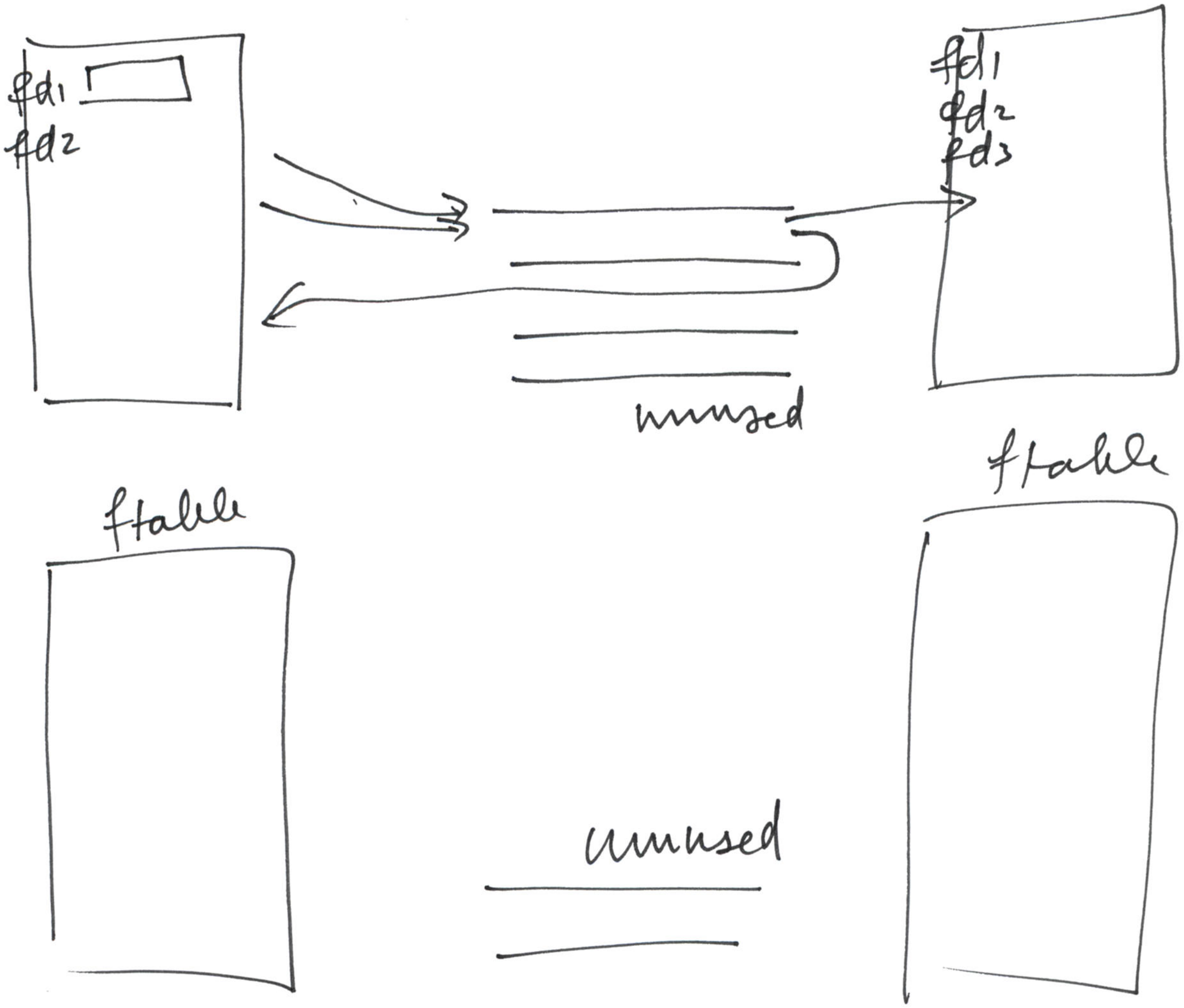
#2 write "Final Exam 2013"

write " Today is Decem "

Parent: write(8, " we are done");

child: read(7, ... buf2);

write(1, ...)


fd1
fd2
fd1
fd2
fd3
unused
ftable
ftable
unused

```
include <stdio.h>
include <signal.h>
define MYSIG 43
```

1

```
* signal hanlder evoked by sig 43
  reinstalls after each sig hit, prints number of hits*/

oid getsig(int s)// signal handler

 static int count = 0;
 printf("signal %d again, %dth time \n", s, ++count);
 signal(MYSIG, getsig); //reinstalling sig handler
```

2

```
nt main(void)

 signal(MYSIG, getsig); // install signal handler
```

3

```
 printf("start counting keyboard kills\n");
 while(1) {};
 return 0;
```

if MYSIG happens
call /jump to "getsig" handler

4 kill (pid, signal#);

initiate ~~the~~ signal?

3. (20 points) Signal and handlers

Consider an ECU (embedded control unit) with the following data structures and tasks. Write the C code or pseudo code for the system executing on this ECU:

(i) It has a queue of "base tasks" that are aperiodic; define this as BQ in your pseudo code

(ii) It has a "cyclic executive schedule" of period tasks A, B and C; define this as PQ in your pseudo code

(iii) It has a scheduler task that gets interrupted by a **signal number 56** periodically (**every 5ms) by the system timer.**

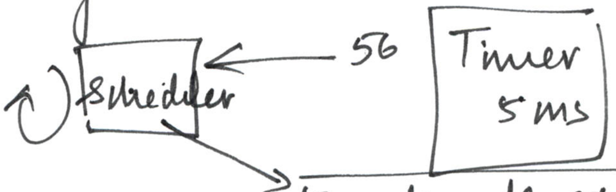
(iv) The scheduler invokes a **signal handler** every time **signal 56** occurs

(v) The signal handler for **signal 56** schedules the next task from the period task queue (PQ). If there is no periodic task then it invokes a base task from the base task queue (BQ).

(vi) After the initiating either task from PQ or BQ, the **signal 56** is armed (set up) again for receiving the signal.

Write the pseudo code for the scheduler including the signal handler, signal setup (arming the signal), timer and the related data structures. You may use dummy functions wherever needed.

```
queee BQ ;      // aperiodic tasks
queeu PQ ;      // cyclic executive of A, B, C
task scheduler ;
```



```
SignalHandler56
task=PQ . deque ( )
if task is null or
   is (PQ is empty;)
   task =BQ . deque ( )
task. execute ( );
   signal (56, ~~Scheduler~~ signalHandler56);
```

~~Ans~~

```
queen BQ;
queue PQ;
→ task timer (5ms)// periodic
task scheduler;
~~tas~~
~~fn~~
scheduler
{ signal (56, signalHS56);
  While (1)
  {
     do something;
  }
}
```

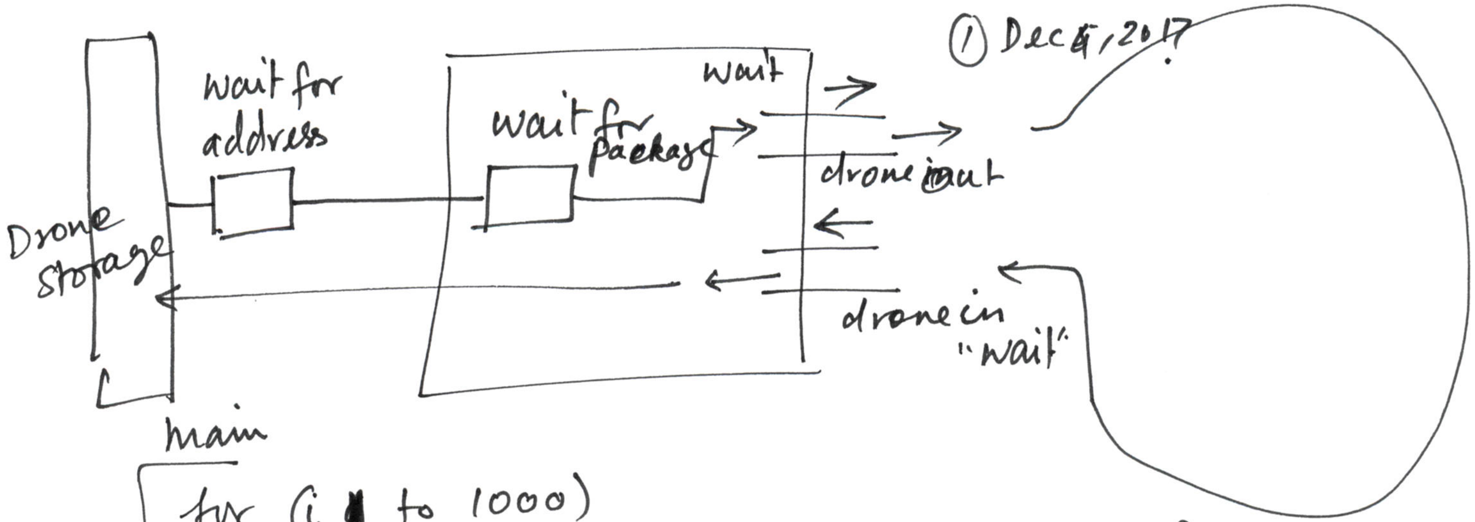
5

```
void signalHandler56 ( )
{ if (! PQ.empty())
     task = PQ.deque();
  else
     ~~task =~~
     if (! BQ. empty())
        task = BQ. ~~deque~~deque();
     else task = sometask;
  $ignal (56, signalHandler56);
```

5. (20 points) Mutual Exclusion and Synchronization

Consider the new project revealed by Amzon.com that delivers packages using drones. Lets see if we can help amazon by organizing their fleet of drones. The drones are powered up and ready to go. Each drone has to **wait** to get (i) the **address** and (ii) the **package**. Next they all fly out of a **single drone port** (**droneOut**) out of the amazon warehouse. Thus the drone port is can handle only one drone at any time. Once the package is delivered drones come back home through another port (**droneIn**) that can also handle/allow only one drone at a time. Assume amazon drones are powered by XINU. Use appropriate **xinu semaphores** to bring order to the amazon drone zone. Clearly show the creation, initialization, usage of the semaphores, in a pseudo code for the operation of the drones.
(Hint: you need 4 semaphores some for synchronization and some for mutual exclusion).

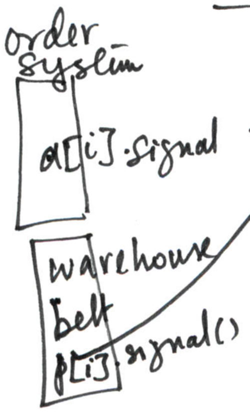
① Dec 4, 2017



main

```
for (i to 1000)
    ready (create ( drone[i]   , deliverPackage, . . . .
    semaphore  a[i] = newsem(0); // synch
    semaphore  p[i] = newsem(0); // synch
semphore  droneOut  = newsem(1); // mutex
semaphore droneIn   = newsem(1); // mutex
```

deliverPackage : "i" drone



```
a[i]. wait( );  // waiting for address
p[i] . wait( ) ; // waiting for package

droneout. wait( ) ;   // waiting for outport to be f
    // get out of the port
droneout. signal ( );
      // go about the zone and deliver pack

dronein . wait( );  // wait for input port
        // go thru' port
dronein. signal ( ); //shelf yourself
 a [i]. signal ( ); // ready to take
                        new address
 p [i]. signal ( );

  :
  :
}
```

(8) Dec 4, 2017

```
deliverPackage: " i " drone

    wait (a[i]);   // waiting for address
    wait (p[i]);

        ~~drone.wait~~
        wait (droneOut);
            // get out of the port
        signal (droneOut);

        // deliver package

        wait (droneIn);
            // get in
        Signal ~~wait~~ (droneIn);
            ~~// go in~~
          // shelf - store yourself drone
          // ready to ~~take~~ next order
          signal (a[i]);
          Signal (P[i]);

    ; // go to sleep;
    }
```