exercises

partial 2016

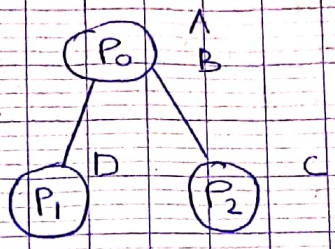
II.

```
1. Void main(){
2.     int p[2], x=1;
3.     pipe(p);
4. * if(fork()){
5.         printf("A\n");
6.     if(fork()){
7.         // wait(0);   (b)
8.         // wait(0);   (b)
9.         printf("B\n");
10.                    }
11.    else {
12.
13.        printf("C\n");
14.                    }
15.            }
16. * else {
17.
18.        printf("D\n");
19.
20.            }
21.    }
```

FABER-CASTELL

a) what are the possible outputs ?

process \ output	
P_0	A
	B
P_1	D
P_2	C



①	②	③	④	⑤	⑥	⑦	⑧
A	A	A	A	A	A	D	D
B	B	C	C	D	D	A	A
C	D	D	B	C	B	B	C
D	C	B	D	B	C	C	B

(b) add (wait(0),) on Line 7, 8
what are the possible outputs now ?

①	②	③
A	A	D
C	D	A
D	C	C
B	B	B

c) we change the lines to :
Line 7: write (P[1], &x, sizeof(int)),
Line 8: write (P[1], &x, sizeof(int)),
Line 12: read (P[0], &x, sizeof(int)),
Line 17: read (P[0], &x, sizeof(int)),

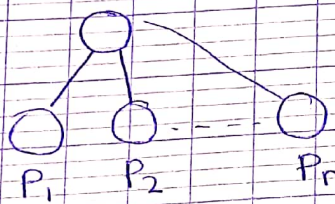
①	②	③	④	⑤	⑥
A	A	A	A	A	A
B	B	C	C	D	D
C	D	D	B	C	B
D	C	B	D	B	C

d) Line 8 is deleted
Line 19 : write (p[1], &x, sizeof(int));

A	A	A	A
B	D	D	C
D	B	C	B
C	C	B	& then will be blocked.

exercise

write a program that creates n-children processes sharing a pipe. Children with even ids → producers of massages time t → print 'a' pipe
children with odd ids → consumers of massages time t → read



this method each process read or write once

```
#define n 100
void main (){
  int p[2], i;
  pipe (p);
  char s = 'a';
  for (i=0; i<n; i++){
    if (!fork ()){
      if (i%2 == 0){
        close (p[0]);
        sleep (2);
        write (p[1], &s, sizeof (char));
      }
```

even id → producers of msg
time t → 'a' pipe
odd id → consumers
time t → read

```
else {
    close (p[1]);
    sleep (2);
    read (p[0], &s, sizeof (char)); }

      break;
            }
```

The quesion is to still read & write

```
void main ( ){
   int p[2], i;
    char s='a' ;
    pipe (p);
   for(i=0 ; i<n ; i++){
      if(!fork())
          break;
                }
   while (1){
     if ( i%2 == 0 && i<N)
          ____
     else if (i<N)
          ____
     else  break;
                }
                }
```

b) Now the producers faster than consumers, and the pipe cant held more than m characters

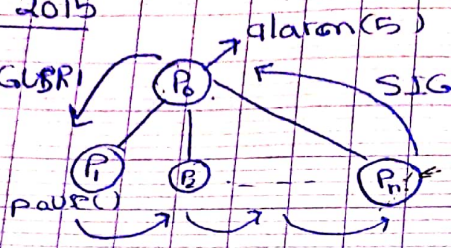
```
void main (){
    int p[2], c[2], i, n, x, m;
    char s='a';
    pipe(p);
    pipe(c);
    scanf("%d",&n);
    scanf("%d",&m);
    for (i=0; i<n; i++){
      if(!fork())
        break;
                }
    while(1){
      if(i%2==0 && i<n){
        close(p[0]);
        read(c[0],&x,sizeof(int));
        if(x<m){
          write(p[1],&s,sizeof(char));
          x++;
          write(c[1],&x,sizeof(int));
                          } }

      else{
        if(i%2!=0 && i<n){
          close(p[1]);
          read(c[0],&x,sizeof(int));
          x--;
          write(c[1],&x,sizeof(int));
          read(p[0],&s,sizeof(char)); }
```

FABER-CASTELL

VC shkeir

1st session 2015

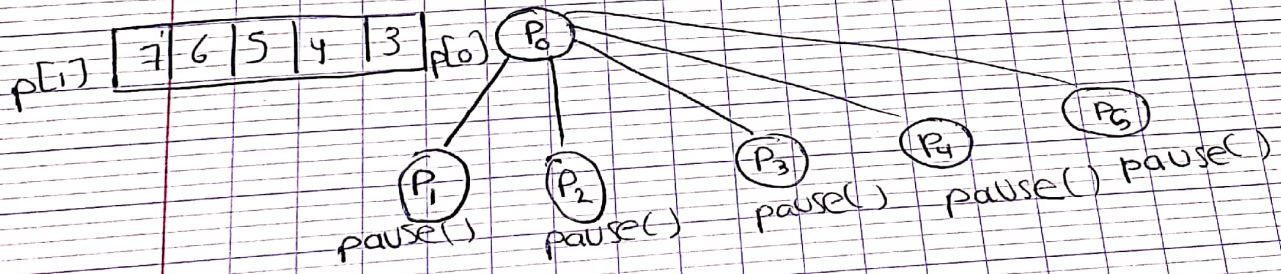
Pb II



pid للخوة
بدي احفظا
بمكان بقدر
access عليه
since each process will send signal to its brother so it must have its pid.

```
void main(){
 int pid,i,p[2];
   pipe(p);
   signal(SIGALARM, handler);
   signal(SIGUSR1, handler);
   signal(SIGUSR2, handler);

  for(i=0;i<n ; i++){
    if(!fork()){
     write(P[1], &getpid(), sizeof(int));
     pause();
     break;
                }  }

if(i==n){
 alarm(5);
 pause();
 read(p[0], &pid, sizeof(int));
 kill(pid, SIGUSR1); pass
 pause();  }
```

```
if (i<n-1){
read(p[0], &pid, sizeof(int));
kill(pid, SIGUSR1); }

else {
        kill(getppid(), SIGUSR2); }
}


void handler(int s){
    if(s==SIGUSR2){
    printf("end");
      exit();
    }
}
```



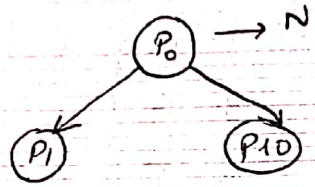
Partial 2014

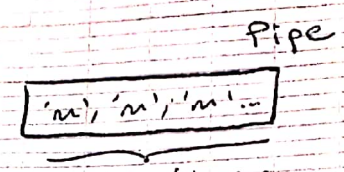
Pb II

```
int count = 0;
void handler (int sig) {
    if (sig == SIGUSR1) {
        exit(count);
    }
}

void main () {
    int st, cpid;
    signal (SIGUSR1, handler);
    signal (SIGUSR2, handler);
    while (1) {
        if (!(cpid = fork ())) {
            break;
        }
        alarm (1);
        pause ();
        kill (cpid, SIGUSR1);
        wait (&st);
        count = WEXITSTATUS(st);
        printf ("the child %d made %d children", cpid, co
    }
    while (1) {
        if (!fork ())
            break;
        count++;
    }
}
```

Nov. 20 2014

Pb III.





- The 10 children starts reading from pipe

```
#define N 10
void handler (int sig) { }

void main () {
  int i, n, p[2], Ecode, mcpid=0, ncpid=0, Pids[10];
  char c = 'm';
  pipe(p);
  signal (SIGUSR1, handler);
  for(i=0; i<N; i++)
    write (p[1], &c, sizeof(char));
  c = '*';
  for(i=0; i<N; i++){
    write (p[1], &c, sizeof(char));
    if(!(Pids[i]= fork())){
      pause();
      break;
    }
}

for(i=0; i<N; i++)
  kill (Pids[i], SIGUSR1);
```

cpid,

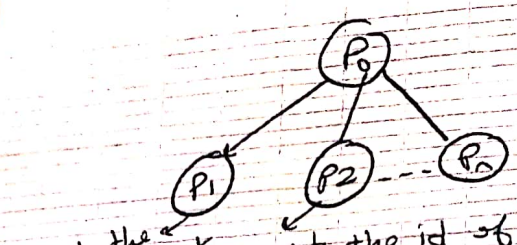
or another way that close the writing then the reading will exit the pipe whenever it is empty, stops when empty

```
if (i == N){
    while ((cpid = wait(&Ecode)) != -1){
        n = WEXITSTATUS (Ecode);
        if (n > ncpid){
            ncpid = n;
            mcpid = cpid;
        }
    }
    printf ("The winner is %d", mcpid);
}
else{
    read (P[0], &c, sizeof(char));
    i=0;
    while (c != '*'){
        read (P[0], &c, sizeof(char));
        i++;
    }
    exit (i);
}
}
```

2012

Pb II.

Part 1

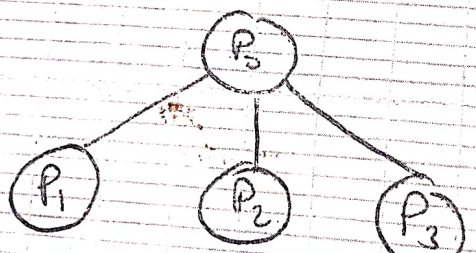


each process prints the pid of the previous process

no pipe

```
#define N ...
void main(){
  int i, prvpid, curPid = getpid();
  for(i=0; i<N; i++){
    PrvPid = CurPid;
    if(!(CurPid = fork())){
      printf("The prev. pid = %d", PrvPid);
      exit(0);
    }
  }
}
```

Part 2



while(1) A1(); while(1) A2(); while(1) A3();

a) Max 2 processes can execute together, just pipe is allow

```c
void main () {
    int i, p[2], n=1;
    pipe (p);
    write( p[1], &n, sizeof(int));
    write ( p[1], &n, sizeof(int));
    for(i=0; i<3; i++){
        if(!fork())
            break;
    }
    if(i==0) {
        while(1) {
            read ( p[0], &n, sizeof(int));
            A1();
            write (p[1], &n, sizeof(int));
        }
    }
    if(i==1) {
        while(1){
            read (p[0], &n, sizeof(int));
            A2();
            write (p[1], &n, sizeof(int));
        }
    }
    if(i== 2) {
        while (1) {
            read(p[0], &n, sizeof(int));
            A3();
            write(p[1], &n, sizeof(int));
        }
    }
}
```

we have 2 processes
If I want only one process to go & other blocked, then 2nd execute, 1st blocked, the 1st execute, 1st blocked.
let the blocked process read from empty pipe
Then to resume it, 1st write in empty pipe

2015

partial

Pb

Pb I

A) Possible Output

Program 1

```
void main(){
int i,j=1,p[2];
pipe(p);
write(p[1],&j,sizeof(int));
for(i=0;i<5;i++){
  if(!fork()){
    close(p[1]);
    break;
  }
}
read(p[0],&j,sizeof(int));
printf("%d\n",i);
}
```

Program 2

```
void main(){
int i,j=1,p[2];
pipe(p);
write(p[1],&j,sizeof(int));
for(i=1;i<5;i++){
  if(!fork()){
    close(p[1]);
    break;
  }
}
wait(0);
read(p[0],&j,sizeof(int));
printf("%d\n",i);
}
```

case I Prog. 1

The parent starts & finishes before any child

5

1
2
3
4 } this permutates

The children aren't blocked, since parent is finished & all children closed their writing, so no more writing side.

Case II (Prog.1)

If any child read from the pipe, so the program will block since all children want to read, but the parent didn't close the writing side.

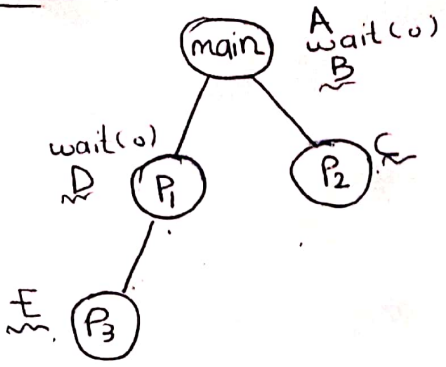
NOV. 21

LAB Session

create a clock 00:00:00

2018-2019

Ex I



1)

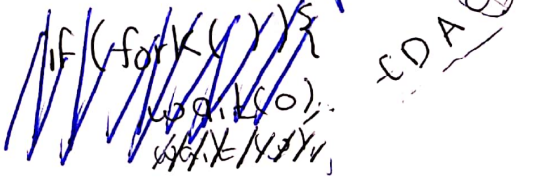
A C B E D
E D A C B
E A C B D
E A C B D
E A D C B

2)

EDCBA

immpossible

CDA



2018 ex1 ②

Ex II

f(2)

j=2