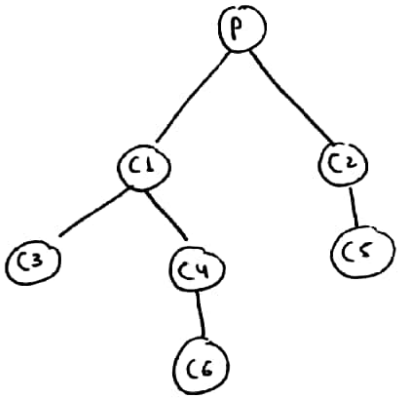
Operating System II: 2016

Problem I:

```
void main () {
    if ((fork() || fork()) && ! (fork())) {
        fork();
    }
}
```



Problem II.

1) ABCD ✓
ACBD ✓
ACDB ✓
BABC
DACB
APBC
ADCB
ABDC ✓

2) ACDB ✓
ADCB ✓
DACB ✓

3) ABCD ✓
ABDC ✓
ADBC ✓
ADCB ✓
ACBD ✓
ACDB ✓

4)
ABDC ✓
ADBC ✓
ADCB ✓
ABC ✓
ACB ✓

Problem III :

1)
```
void main() {
    int p[2], i;
    char c = 'A';
    pipe(p);
    for (i=0; i<N; i++)
        if (!fork())
            break;
    while (1) {
        if (i<N && i%2==0) {
            sleep(2);
            write(p[1], &c, 1);
        }
        else if (i<N && i%2!=0) {
            sleep(2);
            read(p[0], &c, 1);
        }
    }
}
```

2)
```
void main() {
    int p1[2], p2[2], n=0; char c='a'; int i;
    pipe(p1);
    pipe(p2);
    for (i=0; i<N; i++)
        if (!fork())
            break;
    while (1) {
        if (i<N && i%2==0) {
            close(p1[0]);
            read(p2[0], &n, sizeof(int));
            if (n<M) {
                write(p1[1], &c, 1);
                n++;
                write(p2[1], &n, sizeof(int));
            }
            else {
                write(p2[1], &n, sizeof(int));
                sleep(1);
            }
        }
        else if (i<N && i%2!=0) {
            sleep(2);
            close(p1[1]);
            read(p1[0], &c, 1);
            read(p2[0], &n, sizeof(int));
            n--;
            write(p2[1], &n, sizeof(int));
        }
    }
}
```

Operating System II . 2015

Problem I :

A) **Program 1 :**

① If the parent reads first , then all child will print also because they will read from an empty pipe but with closed write side.

possible outputs: 5 (4 3 2 1)!

② If any child reads first, it prints the value of i , but all other processes including the parent will block because they will attempt to read from an empty pipe with opened write side.

possible outputs: 1
2
3
4

Program 2:

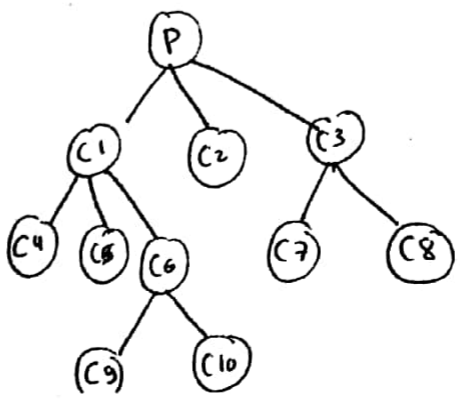
Here the parent cannot read immediately, he must wait at least one child to read from the pipe and exit.

Now the parent and other child processes will attempt to read from an empty pipe with opened write side so they will block.

possible outputs: 1
2
3
4

B) `(fork() || fork()) && fork() && (fork() || (fork() && fork()));`

```
sleep(2);
return 0;
```



Problem III:

Problem II

```
void main () {
    int i, pid;
    int p[2];
    pipe (p);
    signal ( SIGUSR1, handler);
    signal ( SIGUSR2, handler);
    signal ( SIGALRM, handler);
    for (i=1; i<= N; i++) {
        pid = fork();
        if (!pid) {
            close (p[1]);
            pause();
        }
        else
            write (p[1], &pid, sizeof(int));
    }
    close (p[1]);
    alarm (5);
    pause();
}
```

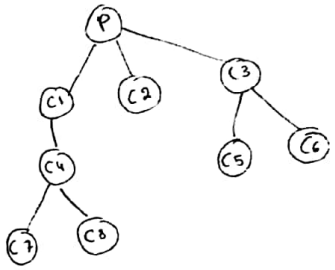
```
void handler ( int nsig) {
    if (nsig == SIGALRM) {
        read ( fd[0], &pid, sizeof(int));
        kill ( pid, SIGUSR1);
        pause();
    }
    else if (nsig == SIGUSR2)
        printf(" End of execution");
    else
        if (read(p[0], &pid, sizeof(int))) {
            kill (pid, SIGUSR2);
            exit();
        }
        else {
            kill (pid, SIGUSR2);
            exit();
        }
}
```

```
..., i<N, i++)
    if (max < WEXITSTATUS ( status[i])) {
        max = WEXITSTATUS (status[i]);
        winner = pid[i];
    }
printf (" the child with pid %d is the winner, it reads %
    characters \n", winner, max);
}
```

Operating System II 2014

Problem I.

```
if (fork())
    fork() && (fork() || (fork() && fork()));
else
    fork() || (fork() && fork());
sleep(5);
```



Problem II:

```
void handler (int signum) {
    if (signum == SIGUSR1)
        exit( counter);
}
```

```
int counter = 0;
void main () {
    int pid, status;
    while (1) {
        if (1 (pid = fork ())) {
            signal (SIGUSR1, handler);
            while (1) {
                if (! fork ())
                    exit ();
                else
                    counter ++;
            }
        }
        else {
            sleep (10);
            kill (pid, SIGUSR1);
            wait (&status);
            printf (" My child %d was created %d processes", pid,
                    WEXITSTATUS (status));
        }
    } // end while
}
```

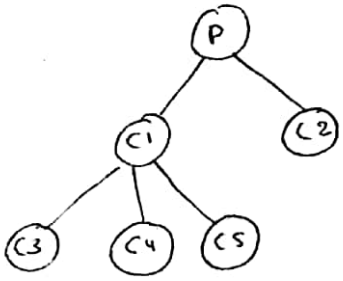
Problem III:

```
int pid[N], status[N], counter=0;
void action (int signum) {
    printf (" captire of SIGUSR1 %d\n", getpid());
}
void main () {
    int fd[2], char c; int n, count=0, j, max, winner;
    pipe (fd);
    for (i=0; i<N; i++) {
        if ((pid[i] = fork()) == 0) {
            signal (SIGUSR1, action);
            close (fd[1]);
            pause ();
            while (nread = read fd[0], &c, sizeof(c))
                count ++;
            sleep (100);
            exit (count);
        }
    }
    close (fd[0]);
    scanf ("%d\n", &n);
    for (j=0; j<n; j++)
        write (fd[1], &c, sizeof (c) +1);
    for (j=0; j<n; j++)
        kill (pid[j] SIGUSR1);
    for (i=0; i<N; i++)
        waitpid (pid[i], &status [i])
    max = WEXITSTATUS (status [0]);
```

Operating System II : 2013

Problem I

1)
```
void main () {
    if (fork() || (fork() && fork())) {
        fork();
    }
}
```



2) ACBD
ADBC
ACDB
ADCB

CABD
CDAB
CADB

DABC
DACB
DCAB

Problem II :

1)
```
void main() {
    int i;
    for( i=0; i<N; i++)
        if (!fork() )
            break;
    if (i == N)
        while (wait(NULL));
    else if (i%2 == 0)
            FE();
        else
            FO();
}
```

2)
```
if (i==N) {
    write ( fd[i], &token, sizeof(int));
    while (wait(NULL));
}
else {
    read (fd[i-1], &token, sizeof(int));
    if (i%2==0)
        FE();
    else
        Fo();
    write (fd[i], &token, sizeof(int));
}
```

3)
```
if (i==N) {
    write (fd[i], &token, sizeof(int));
    while (wait(NULL));
}
else if (i%2==0) {
    while (Flag) {
        read (fd[0], &token, sizeof(int));
        if (token==0) {
            Flag=0;
            FE();
        }
        write (fd[i], &token, sizeof(int));}
    }
else {
    read ( fd[0], &token, sizeof(int));
    token++;
    write (fd[i], &token, sizeof(int));
    Fo();
    read (fd[0], &token, sizeof(int));
    token--;
    write ( fd[i], &token, sizeof(int));
}
```

1) Write a program that creates N child [illegible] the following rules:

- The child process with even PID executes the function [illegible]
- The child process with odd PID executes the function [illegible]
- The parent must wait the termination of all children [illegible]

2) Now, suppose that all child processes shared a file and the child (odd and even) use the functions FE() and Fo() for accessing the file. We assume that in a given instant, one process (even or odd) can write in the file (the other processes, wanting to write to the file, must be blocked until the end of the current writing process).

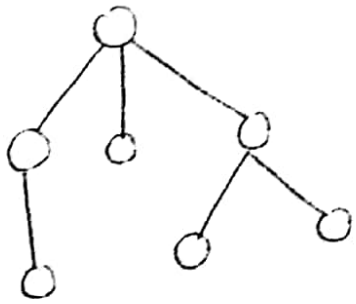
Using pipes of communications, add few lines to the code of part (I) in order to achieve the above described synchronization.

3) In this part, we suppose that the even children are writers and the odd children are readers. If a writer process is writing to a file using FE(), we must prevent all other processes to access the file. In contrast, several readers can read simultaneously the file using Fo without any writer. Using pipes perform the described synchronization

Operating System II : 2012

Problem I:

1)
```
if (fork())
    if (fork())
        if(! fork())
            if (fork())
                fork();
    else
        fork();
```



2)
```
if (fork())
    fork() && (fork() || (fork() && fork()));
else
    fork();
```

Problem II:

```
int h=0, m=0, s=0;

void activate (int sig) {
    s++;
    if (s==60) {
        m++;
        s=0;
    }
    if (m==60) {
        h++;
        m=0;
    }
    if (h==24)
        h=0;
}

void main () {
    signal (SIGALRM, activate);
    while (1) {
        printf ("\n%c%c:%c%c:%c%c\n",
        h/10+'0', h%10+'0', m/10+'0', m%10+
        '0', s/10+'0', s%10+'0');

        alarm(1);
        pause();
    }
}
```

Problem III.

1) ABCD
AC DB
ACBD
CDAB
CABD
CADB

2)
```
void main () {
    if (fork()) {
        wait (NULL),
        printf ("A"),
        printf ("B"),
    }
    else {
        printf ("C"),
        printf ("D"),
    }
```

```
void handler (int signum) {
    printf ("C"),
}

void handler 2 (int signum) {
    printf ("B");
}
```

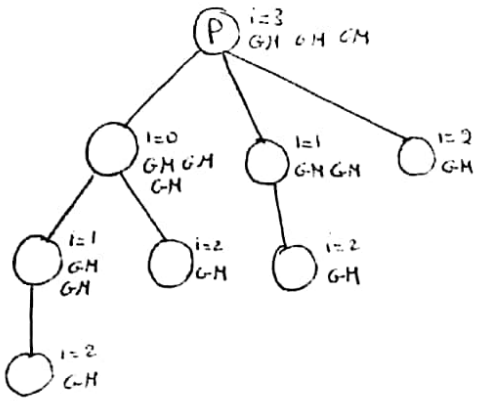
3)
```
int child = 0,
int father = 0,

void main () {
    int pid;
    signal (SIGUSR1, handler 1),
    signal (SIGUSR2, handler 2),
    if (pid = fork()) {
        father = getpid(),
        printf ("A"),
        kill (pid, SIGUSR1),
        pause(),
    }
    else {
        child = getpid(),
        pause(),
        kill (father, SIGUSR2),
        sleep(2),
        printf ("D");
    }
}
```

Operating System II : 2011

Problem I :

Program 1 .



i4 "Good Morning"