Lecture 3

GO CLASSES

initial few lectures plan

- How to create a process ?
- System call / interrupts
- what happens when we switch on the computer
- User mode - Kernel mode
- threads (user level / Kernel level / many to one)
- Context switch of processes
- Context switch of threads

GO CLASSES

initial few lectures plan

- How to create a process ?
- System call / interrupts
- what happens when we switch on the computer
- User mode - Kernel mode
- threads (user level / kernel level / many to one)
- Context switch of processes
- Context switch of threads

Till lecture 6.

after all of the topics listed in previous slide, we will start

→ Scheduling.

at lecture 7

GO CLASSES

Objective :

understand how does the 2 steps get perform.

fork-exec
1. Create space for hello.out
2. load hello.out in that space

→ understand fork

→ understand exec

→ OC will come back to process creation.

Continue ..

Fork questions

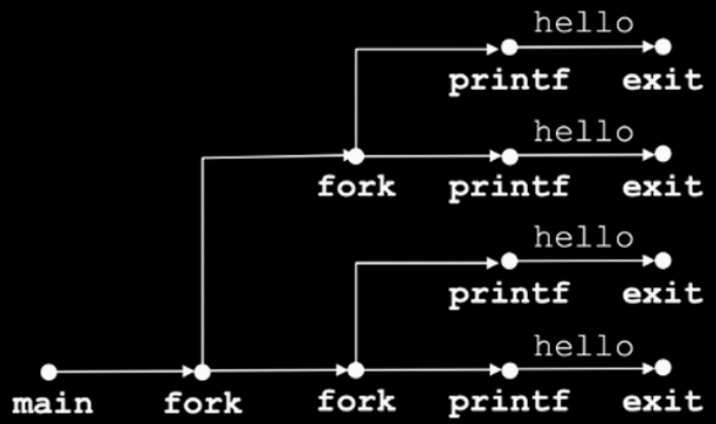
Process graph

Question 1

we did in previous class

```
1 int main()
2 {
3     Fork();
4     Fork();
5     printf("hello\n");
6     exit(0);
7 }
```

```
1 int main()
2 {
3     Fork();
4     Fork();
5     printf("hello\n");
6     exit(0);
7 }
```



Question 2

1. How many Hello's are printed. (Assume fork does not fail.)

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
    printf("Hello\n");
  }
  return 0;
}
```

we did in previous class

1. 2
2. 4
3. 6
4. 8

1. How many Hello's are printed. (Assume fork does not fail.)

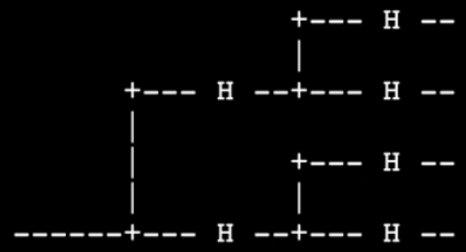
```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
    printf("Hello\n");
  }
  return 0;
}
```

1. 2
2. 4
3. 6 ✓
4. 8

Solution Remarks: This code is equivalent to:

```
fork();
printf("Hello\n");
fork();
printf("Hello\n");
```

Now draw the diagram (where H is for the print statement):



Question 3

How many Hello's are printed. (Assume fork does not fail.)

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
  }
  printf("Hello\n");
  return 0;
}
```

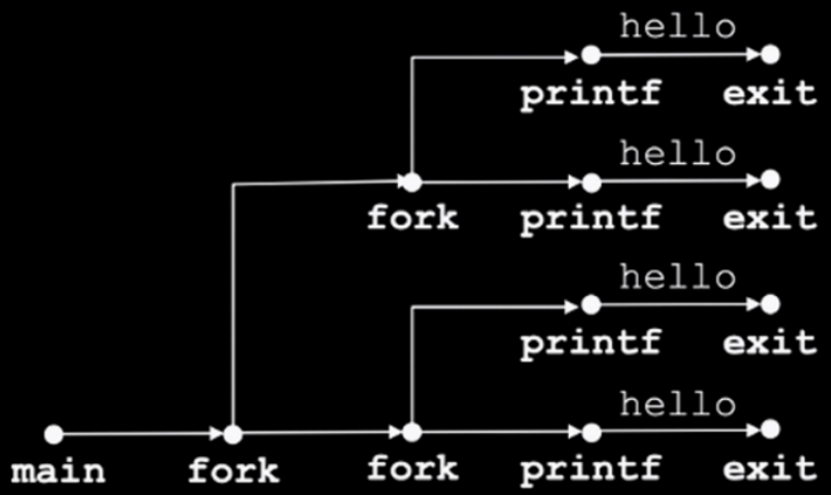
we did in previous class

1. 2
2. 4
3. 6
4. 8

```
int main() {
  int i;
  for(i = 0; i < 2; i++) {
    fork();
  }
  printf("Hello\n");
  return 0;
}
```

This code is equivalent to:

```
Fork();
Fork();
printf("hello\n");
exit(0);
```



Question 4

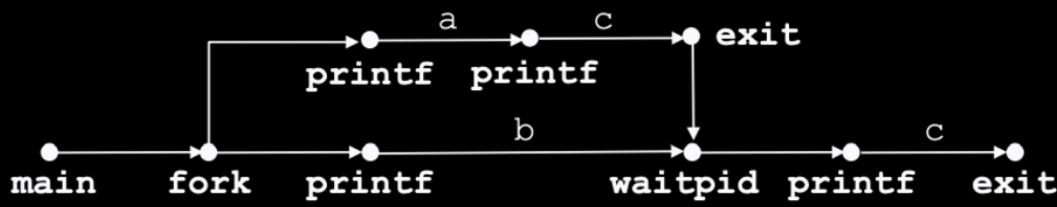
Write down all possible patterns printed by following code.

```
1  int main()
2  {
3      if (Fork() == 0) {
4          printf("a");
5      }
6      else {
7          printf("b");
8          waitpid(-1, NULL, 0);
9      }
10     printf("c");
11     exit(0);
12 }
```

we did in previous class

Write down all possible patterns printed by following code.

```
1  int main()
2  {
3      if (Fork() == 0) {
4          printf("a");
5      }
6      else {
7          printf("b");
8          waitpid(-1, NULL, 0);
9      }
10     printf("c");
11     exit(0);
12 }
```



Question 5

How many child process will create ?

```
for ( i = 1; i<=3; i++ )
{
   fork();
}
```

fork ( )
fork ( )
fork ( )

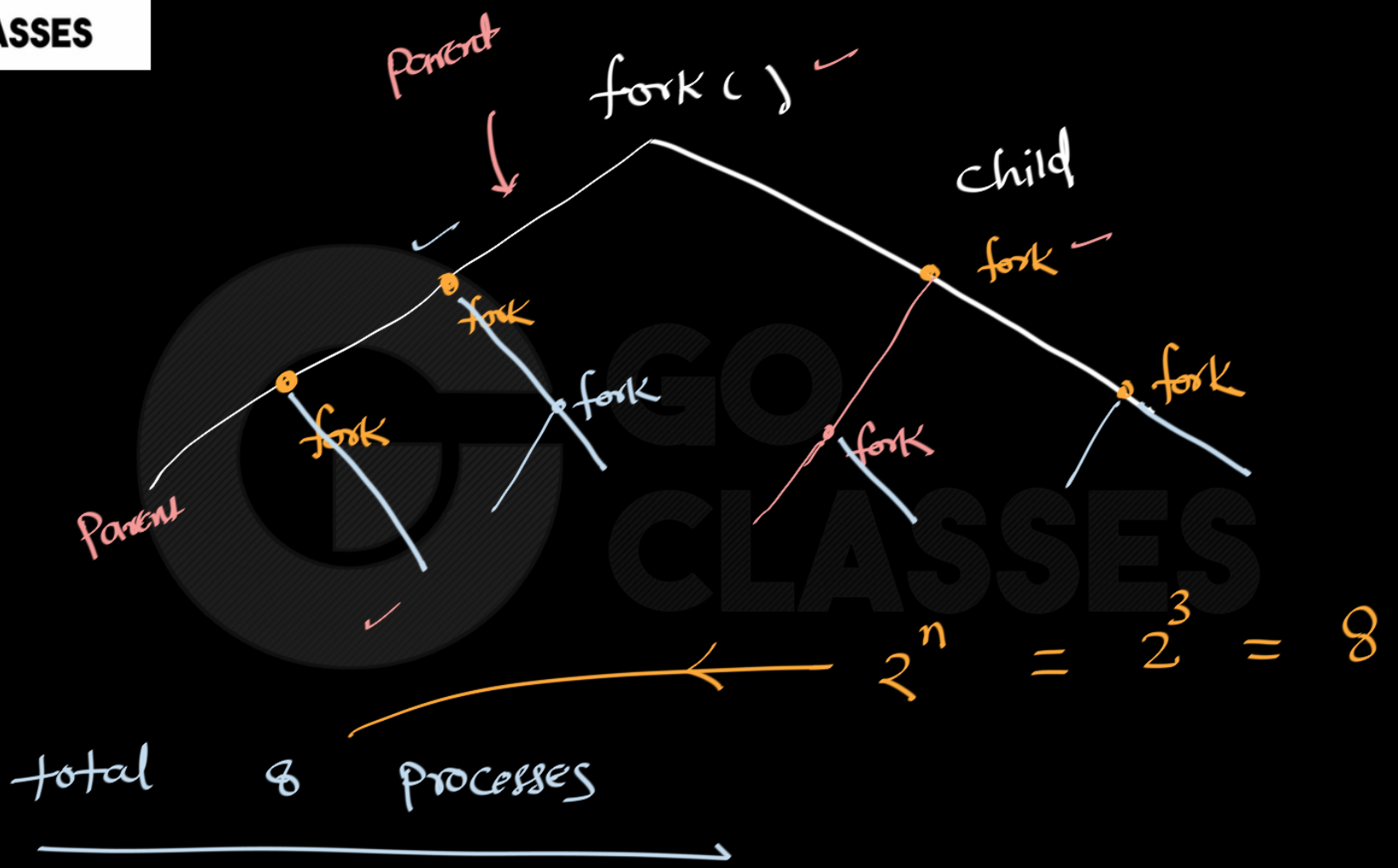
Question 5

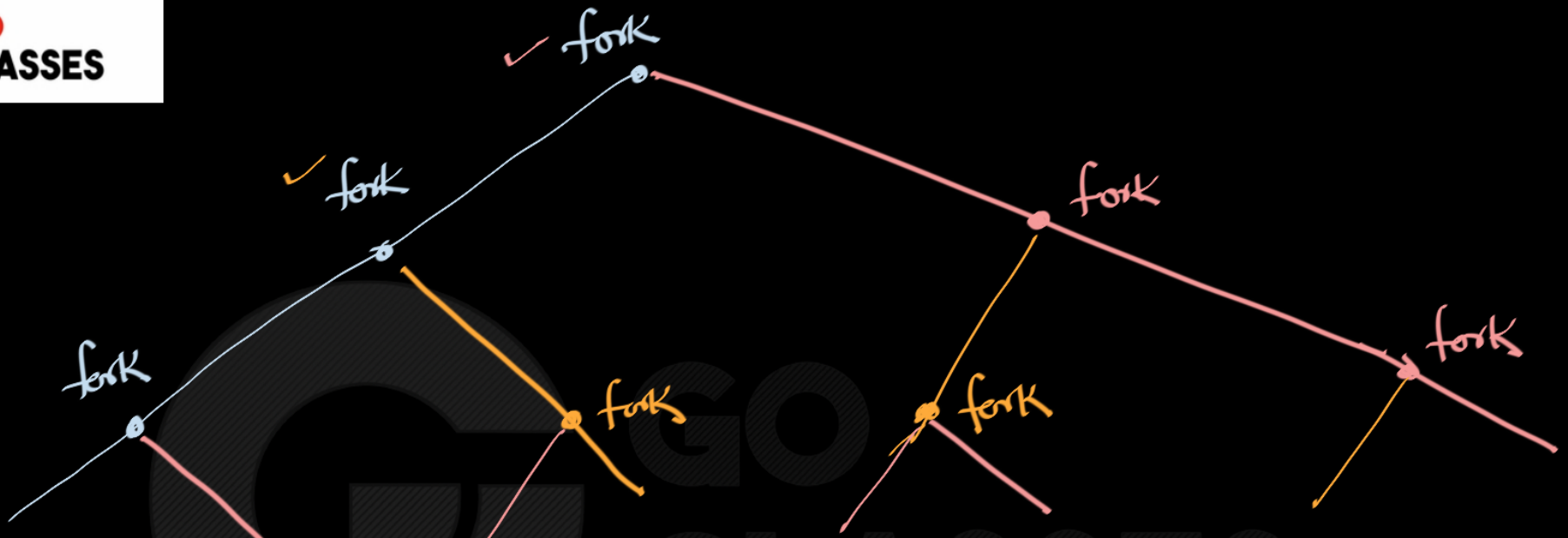
How many child process will create ?

```
for ( i = 1; i<=3; i++ )
{
   fork();
}
```

total processes = 8

Child processes = 8 - 1 = 7


GO CLASSES
Parent
fork ( )
child
fork
fork
fork
fork
fork
fork
Parent
$2^n = 2^3 = 8$
total 8 processes



"fork-tree"

GO CLASSES

fork() ← creates Just one Process

original process is already there hence total Processes after executing fork = 2

Question 6

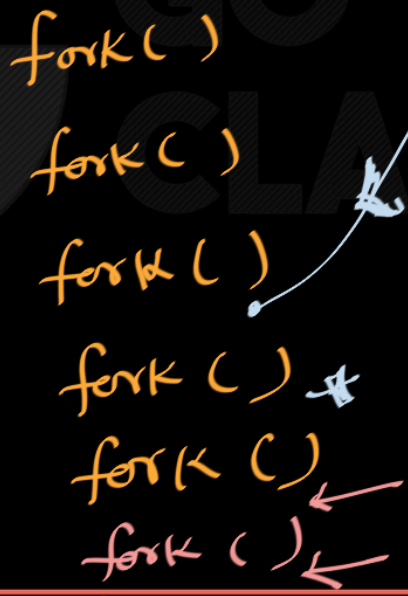
How many child process will create ?

```
for ( i = 1; i<=3; i++ )
{
   fork();
   fork();

}
```

Question 6

How many child process will create ?

```
for ( i = 1; i<=3; i++ )
{
   fork();
   fork();

}
```



total processes = 2^6

Child processes = 63

Question

GATE CSE 2008 | Question: 66

14,032 views

24

A process executes the following code

```
for(i=0; i<n; i++) fork();
```

The total number of child processes created is

A. n
B. $2^n - 1$
C. 2^n
D. $2^{n+1} - 1$

gatecse-2008 operating-system fork-system-call normal

Question

GATE CSE 2008 | Question: 66

14,032 views

24

A process executes the following code

```
for(i=0; i<n; i++) fork();
```

The total number of child processes created is

A. n
B. $2^n - 1$
C. 2^n
D. $2^{n+1} - 1$

gatecse-2008 operating-system fork-system-call normal

Question 8

How many processes are created with these fork() statements?

15

11

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

https://stackoverflow.com/questions/19106576/how-many-processes-are-created-with-these-fork-statements

Question 8

How many processes are created with these fork() statements?

15

11

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

There are 8 processes which will be executing

```c
if (pid == 0)
{
  fork();
}
fork();
return 0;
```

Parent

```c
pid = fork();
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
```

Child

```c
pid = fork();
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
```

https://stackoverflow.com/questions/19106576/how-many-processes-are-created-with-these-fork-statements

Question 8

How many processes are created with these fork() statements?

15

11

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>

int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

after the first fork —

Parent

```c
pid = fork();
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
```

Child

```c
pid = fork();
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
```

https://stackoverflow.com/questions/19106576/how-many-processes-are-created-with-these-fork-statements

GO CLASSES

after the first fork —

Parent

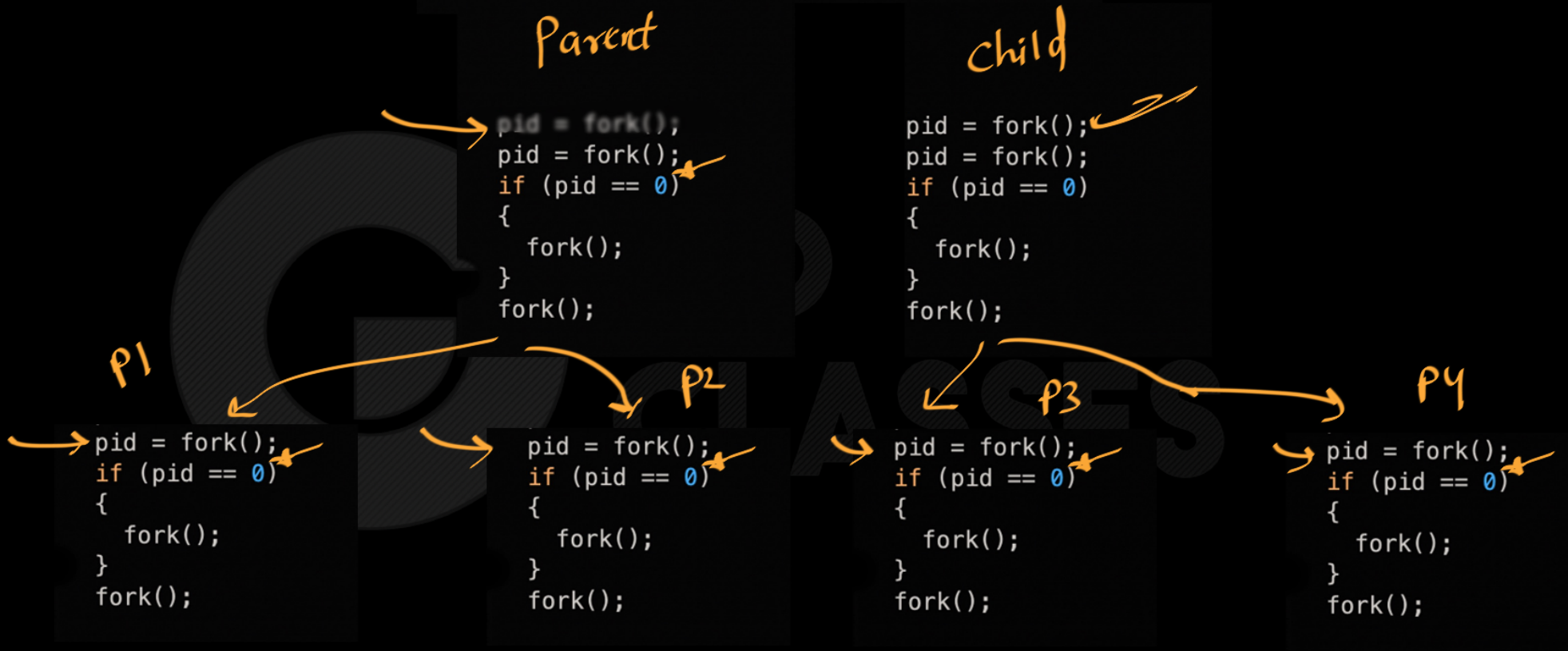
```
pid = fork();
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
```

child

```
pid = fork();
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
```

GO CLASSES

after the first fork –



GO CLASSES

```c
int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

6x4 = 24 Processes

P1

```c
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
return 0;
}
```

P2

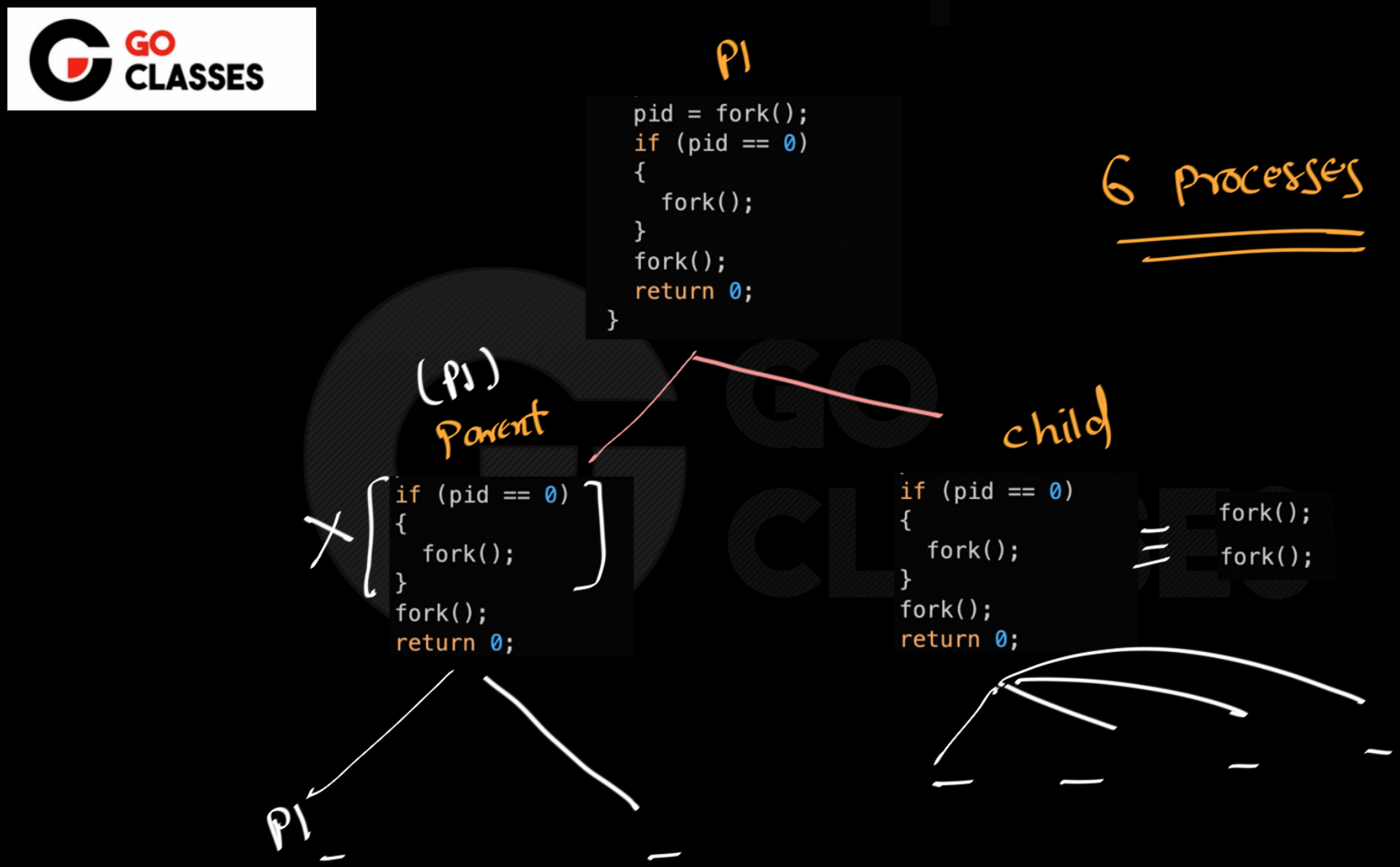
```c
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
return 0;
}
```

P3

```c
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
return 0;
}
```

P4

```c
pid = fork();
if (pid == 0)
{
  fork();
}
fork();
return 0;
}
```


P1
pid = fork();
if (pid == 0)
{
fork();
}
fork();
return 0;
}
6 Processes
(P1)
Parent
child
if (pid == 0)
{
fork();
}
fork();
return 0;
fork();
fork();
P1

GO CLASSES

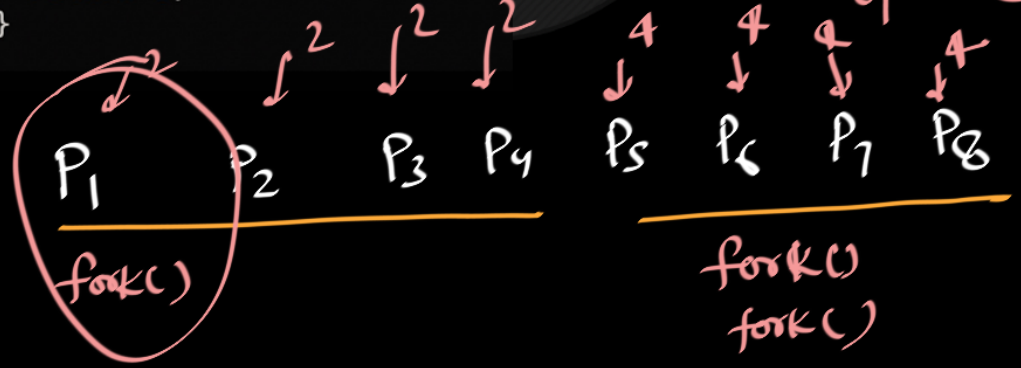
Method 2

$2 \times 4 + 4 \times 4$

$= 24$

```c
int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

we have 8 processes that will execute remaining line of codes




```c
if (pid == 0)
{
  fork();
}
fork();
return 0;
}
```

$8 + 16$

$= 24$

```c
int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

8

P_1 → child of P_1

4 Processes here P_1, P_2, P_3, P_4

GO CLASSES

8 + 16 = 24

```
int main(void)
{
  pid_t pid = fork();
  pid = fork();
  pid = fork();
  if (pid == 0)
  {
    fork();
  }
  fork();
  return 0;
}
```

How many Processes here = 4

P_1, P_2, P_3, P_4

How many Processes here = 8

P_1, P_2, P_3, P_4, child of P_1, child of P_2, child of P_3, child of P_4

4 Processes

fork()

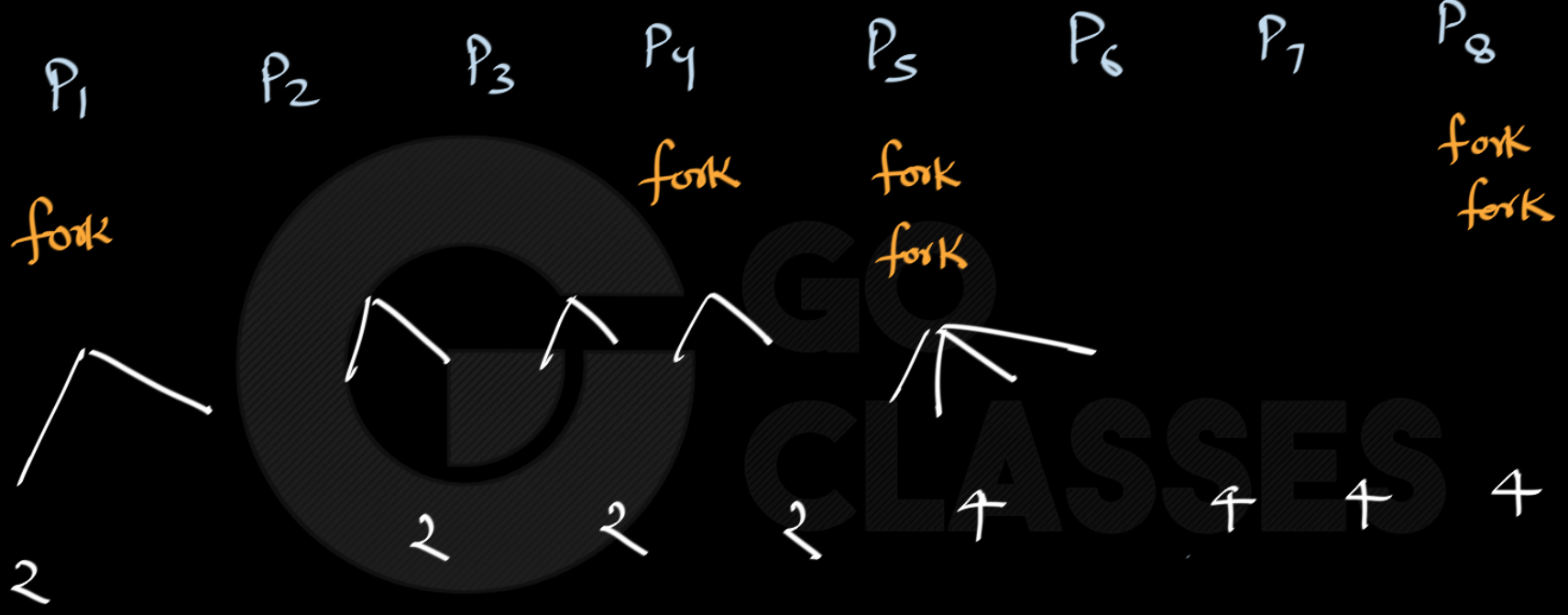
4 Processes

fork()

fork()

Nonzero pid

zero pid



= 24 Answer

It's fairly easy to reason through this. The `fork` call creates an additional process every time that it's executed. The call returns `0` in the new (child) process and the process id of the child (not zero) in the original (parent) process.

```
pid_t pid = fork();  // fork #1
pid = fork();        // fork #2
pid = fork();        // fork #3
if (pid == 0)
{
  fork();            // fork #4
}
fork();              // fork #5
```

1. Fork #1 creates an additional processes. You now have two processes.
2. Fork #2 is executed by two processes, creating two processes, for a total of four.
3. Fork #3 is executed by four processes, creating four processes, for a total of eight. Half of those have `pid==0` and half have `pid != 0`
4. Fork #4 is executed by half of the processes created by fork #3 (so, four of them). This creates four additional processes. You now have twelve processes.
5. Fork #5 is executed by all twelve of the remaining processes, creating twelve more processes; you now have twenty-four.

Question 9

Question I.4 (10 points)

Consider the following example program. List all legal outputs this program may produce when executed on a Unix system. The output consists of strings made up of multiple letters.

```
#include <unistd.h>
#include <sys/wait.h>

// W(A) means write(1, "A", sizeof "A")
#define W(x) write(1, #x, sizeof #x)

int main()
{
    W(A);
    int child = fork();
    W(B);
    if (child)
        wait(NULL);
    W(C);
}
```

https://courses.cs.vt.edu/cs3214/spring2015/html/documents/Midterm-Exam-Sp2015-key.pdf

GO CLASSES

A B C

B

```
int main()
{
    W(A);
    int child = fork();
    W(B);
    if (child)
          wait(NULL);
    W(C);
}
```

Parent

child

```
int child = fork();
W(B);
wait(NULL);
W(C);
```

W(B)

W(C)

total 2 Patterns

A B B C C

A B B C C

A B C B C

Same

A C B B C

Question I.4 (10 points)

Consider the following example program. List all legal outputs this program may produce when executed on a Unix system. The output consists of strings made up of multiple letters.

```
#include <unistd.h>
#include <sys/wait.h>

// W(A) means write(1, "A", sizeof "A")
#define W(x) write(1, #x, sizeof #x)

int main()
{
    W(A);
    int child = fork();
    W(B);
    if (child)
            wait(NULL);
    W(C);
}
```

Answer: Two possible answers: ABBCC and ABCBC

Question 10

13. (5 points) How many times does the following program print "Hello!"?

```
int a = 0;
int rc = fork();
a++;
if (rc == 0) {
  rc = fork();
  a++;
} else {
  a++;
}
printf("Hello!\n");
printf("a is %d\n", a);
```

○ 2 ○ 3 ○ 4 ○ 6 ○ None of the above

14. (5 points) Considering the same program, what will be the largest value of a, displayed by the program?

○ 2 ○ 3 ○ 5 ○ None of the above

GO CLASSES

```
int a = 0;
int rc = fork();
a++;
if (rc == 0) {
  rc = fork();
  a++;
} else {
  a++;
}
printf("Hello!\n");
printf("a is %d\n", a);
```

Parent

```
a++;
if (rc == 0) {
  rc = fork();
  a++;
} else {
  a++;
}
printf("Hello!\n");
printf("a is %d\n", a);
```

Child

```
a++;
if (rc == 0) {
  rc = fork();
  a++;
} else {
  a++;
}
printf("Hello!\n");
printf("a is %d\n", a);
```

```
int a = 0;
int rc = fork();
a++;
if (rc == 0) {
  rc = fork();
  a++;
} else {
  a++;
}
printf("Hello!\n");
printf("a is %d\n", a);
```

Parent

```
a++;
a++;
printf("Hello!\n");
printf("a is %d\n", a);
```

Child

```
a++:
rc = fork();
a++;
printf("Hello!\n");
printf("a is %d\n", a);
```

Three times Hello

1

2

14. (5 points) Considering the same program, what will be the largest value of a, displayed by the program?

○ 2 ○ 3 ○ 5 ○ None of the above

Parent

a=0

```
a++;
a++;
printf("Hello!\n");
printf("a is %d\n", a);
```

3

Child

a=0

a++: ← 1

```
rc = fork();
a++;
printf("Hello!\n");
printf("a is %d\n", a);
```

```
a++;
printf("Hello!\n");
printf("a is %d\n", a);
```

2

```
a++;
printf("Hello!\n");
printf("a is %d\n", a);
```

2

13. B
14. A

Question 11

MSQ

Possible patterns it can print?.

Part (b) [3 MARKS]

```
int main() {
        int i = 0;

        while (i < 2) {
                fork();
                printf("%d ", i);
                i = i + 1;
        }
}
```

A: 0 1 0 1
B: 0 0 1 1 1 1
C: 0 1 0 1 1 1
D: 0 1 1 0 1 1
E: 0 1 1 1 0 1

Question 11

MSQ

Possible patterns it can print?.

Part (b) [3 MARKS]

```
int main() {
        int i = 0;

        while (i < 2) {
                fork();
                printf("%d ", i);
                i = i + 1;
        }
}
```

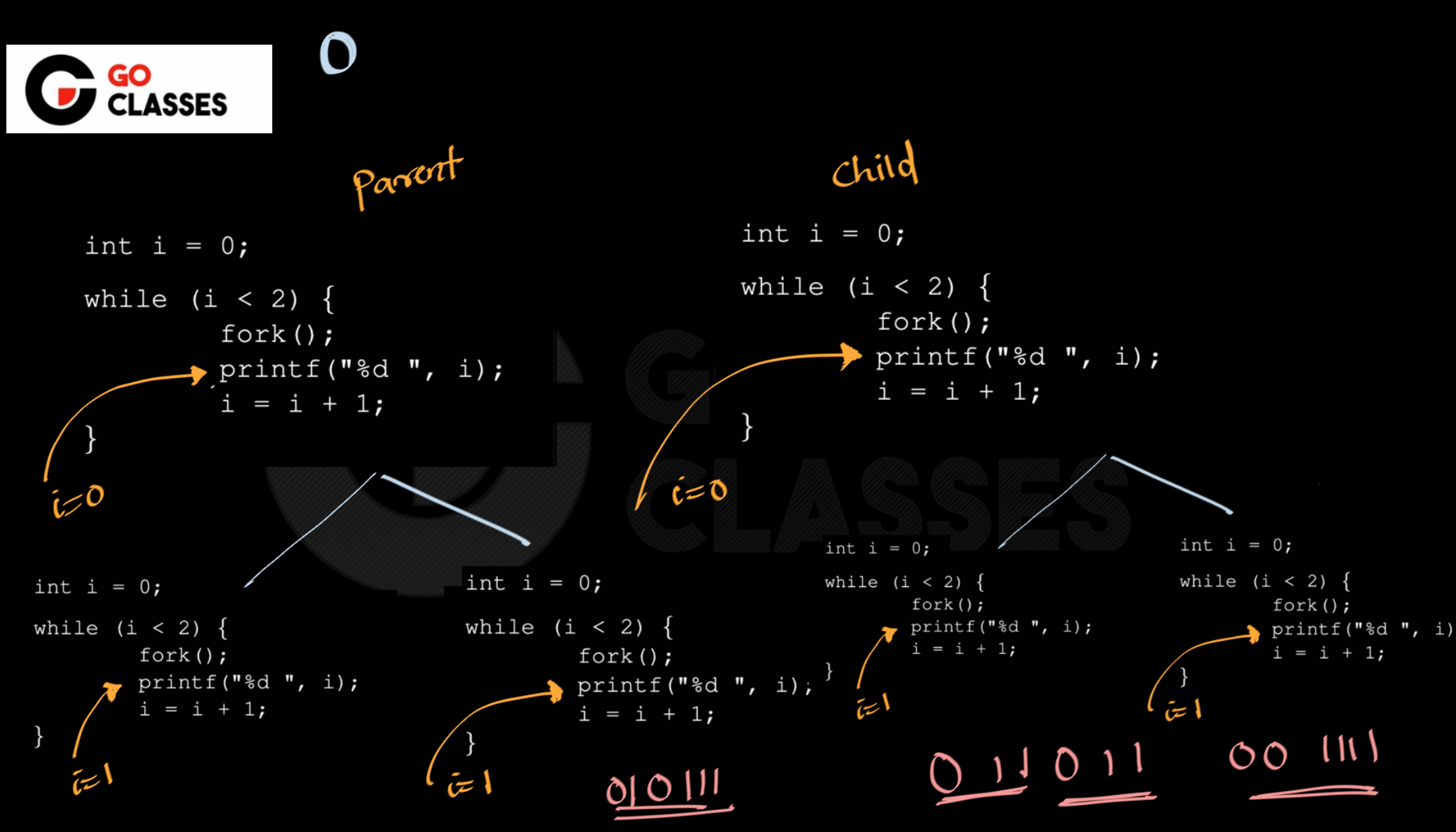
A: 0 1 0 1

B: 0 0 1 1 1 1

C: 0 1 0 1 1 1

D: 0 1 1 0 1 1

E: 0 1 1 1 0 1


GO CLASSES
0
Parent
Child
int i = 0;
while (i < 2) {
fork();
printf("%d ", i);
i = i + 1;
}
i=0
i=1
0 1 0 1 1 1
0 1 0 1 1
0 0 1 1 1

GO CLASSES

Dependency graph

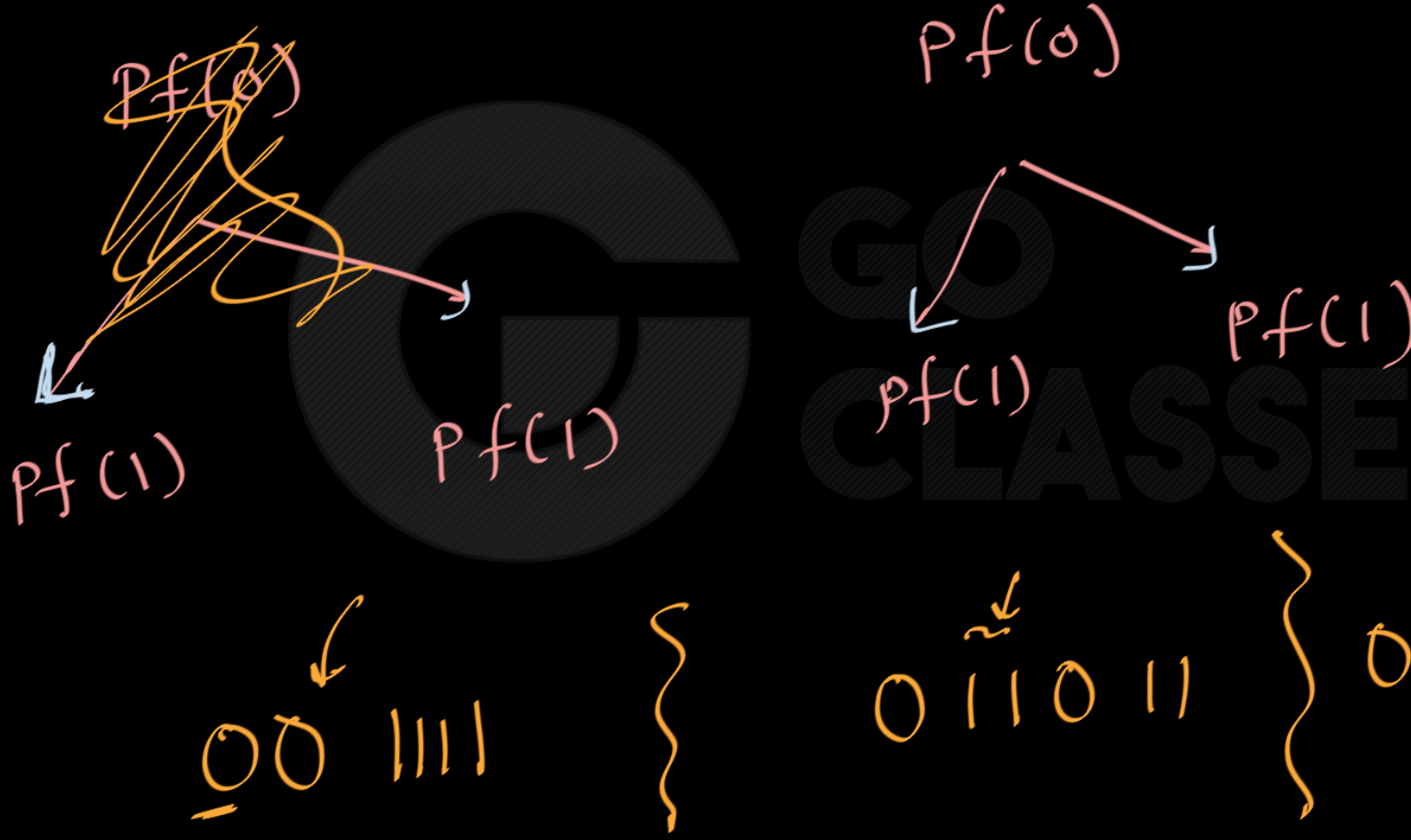
O

0 1

0 1 0

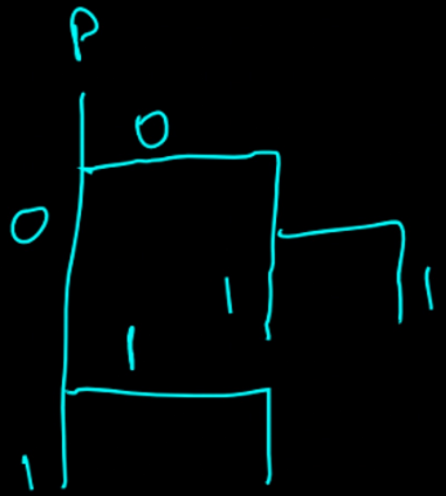
0 1 0 1 1

0 1 0 1 1


Pf(0)
Pf(1)
Pf(1)
Pf(0)
Pf(1)
Pf(1)
00 1111
0110 11
010 111

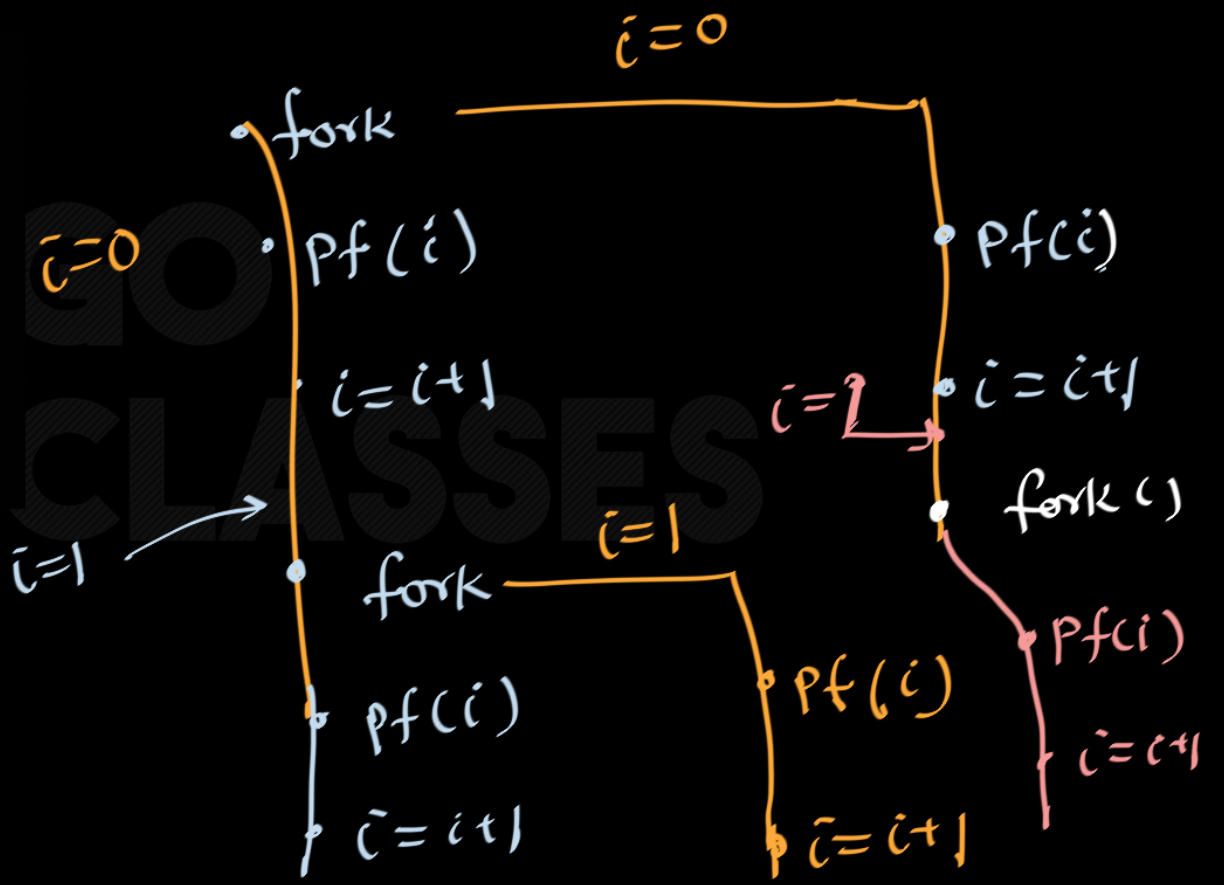
Part (b) [3 MARKS] whatB

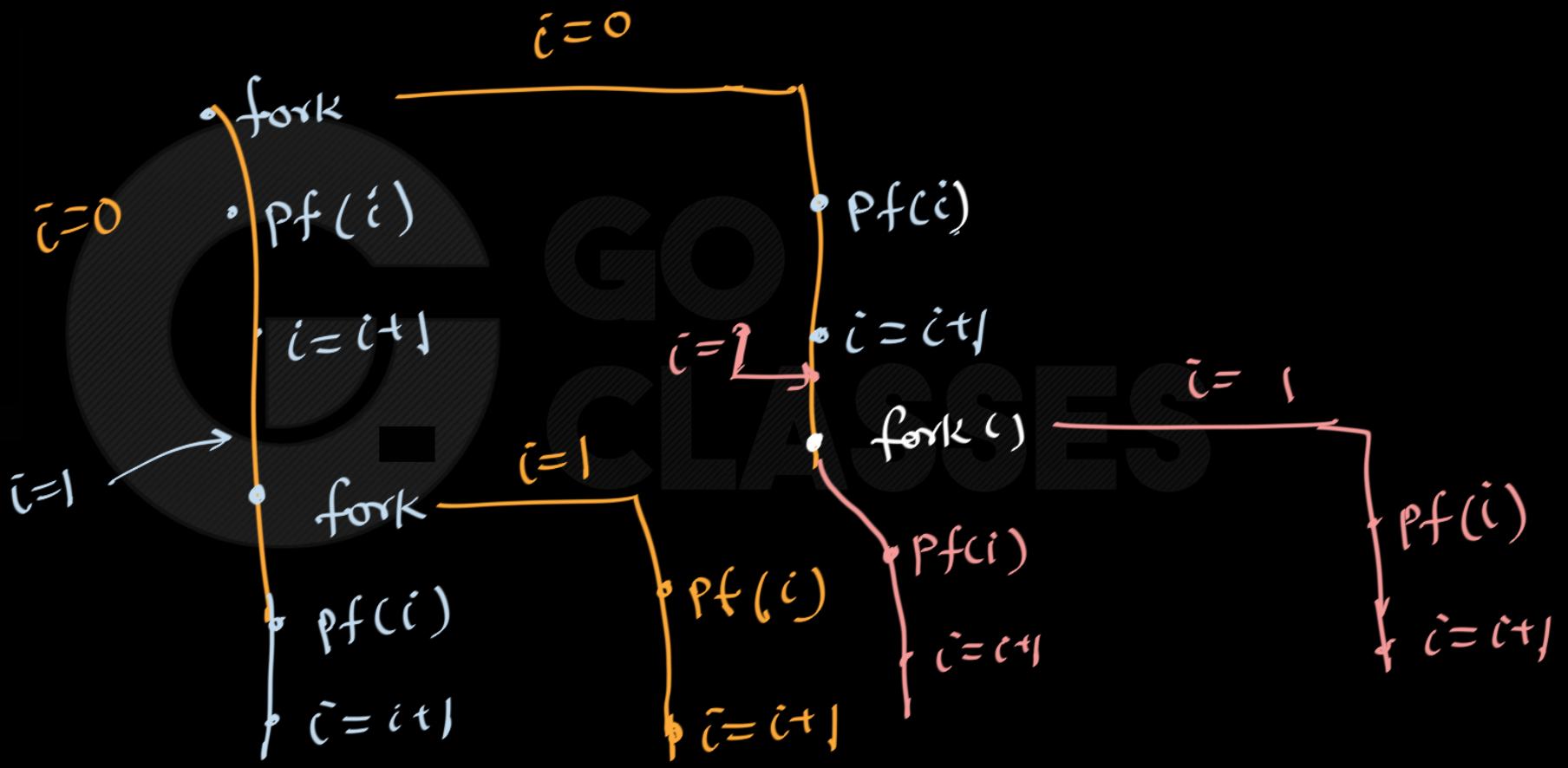
```
int main() {
        int i = 0;

        while (i < 2) {
                fork();
                printf("%d ", i);
                i = i + 1;
        }
}
```

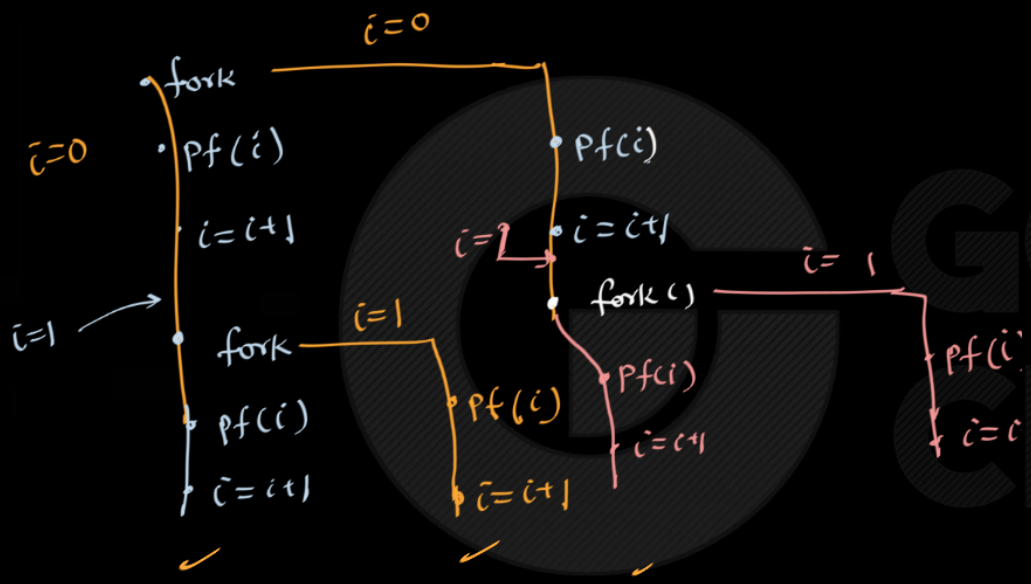


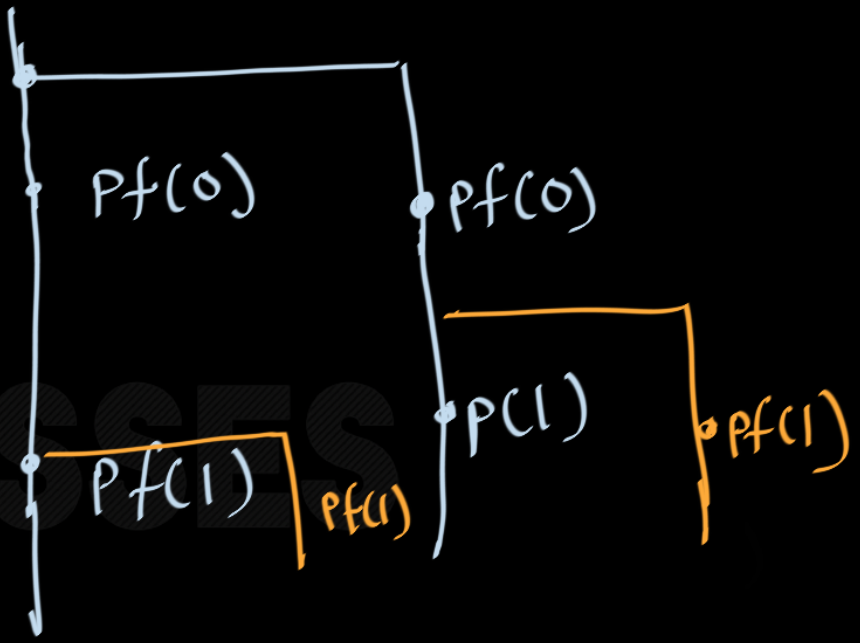
A:	0	1	0	1		
B:	0	0	1	1	1	1
C:	0	1	0	1	1	1
D:	0	1	1	0	1	1
E:	0	1	1	1	0	1

```
int main() {
        int i = 0;

        while (i < 2) {
                fork();
                printf("%d ", i);
                i = i + 1;
        }
}
```




i=0
fork
i=0
pf(i)
i=i+1
pf(i)
i=i+1
i=1
i=1
fork()
i=1
fork
pf(i)
i=i+1
pf(i)
i=i+1
pf(i)
i=i+1
pf(i)
i=i+1


i=0
fork
i=0
pf(i)
i=i+1
i=1
fork
i=1
pf(i)
i=i+1
pf(i)
i=1
pf(i)
i=i+1
fork()
i=1
pf(i)
i=i


Pf(0)
Pf(1)
Pf(1)
Pf(0)
P(1)
Pf(1)

Question 12

Part (c) [3 MARKS] whatC

```
int main() {
        int i = 0;

        while (i < 2) {
                int pid = fork();
                if (pid)
                        waitpid(pid, NULL, 0);
                printf("%d ", i);
                i = i + 1;
        }
}
```

H.W.

A:	0	1	0	1		
B:	0	0	1	1	1	1
C:	0	1	0	1	1	1
D:	0	1	1	0	1	1
E:	0	1	1	1	0	1

Part (c) [3 MARKS] whatC

```
int main() {
        int i = 0;

        while (i < 2) {
                int pid = fork();
                if (pid)
                        waitpid(pid, NULL, 0);
                printf("%d ", i);
                i = i + 1;
        }
}
```

A: 0 1 0 1
B: 0 0 1 1 1 1
C: 0 1 0 1 1 1
D: 0 1 1 0 1 1
E: 0 1 1 1 0 1

---

http://www.exams.skule.ca/exams/ECE344H1_20189_6315668516943-processes.pdf

GO CLASSES

Question 13

Very tough

H.W.

Question 5. Fork Me [10 MARKS]

```
int main()
{
    int c = 0;
    for (int i = 0; i < 3; i++) {
        if (fork() == 0) {
            printf("%d_", c + i);
            fflush(stdout);
        } else {
            c = c + i + 3;
        }
    }
    return 0;
}
```

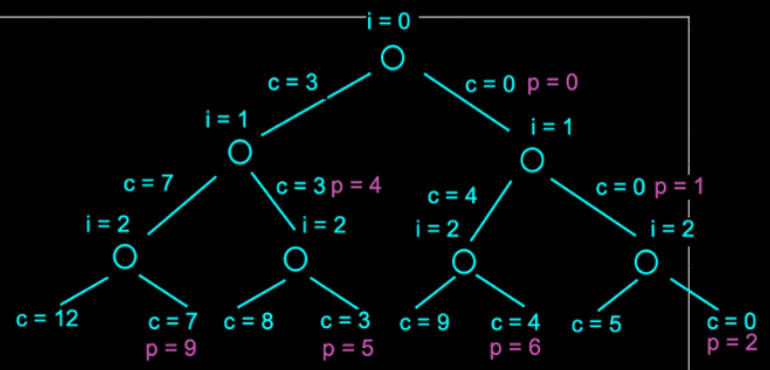
Part (a) [2 MARKS] In the code shown above, how many times is `fork()` invoked?

Part (b) [4 MARKS] Circle all the output sequences shown below that are valid program output. Marks will be deducted for incorrect answers. Hint: Make a diagram that shows program execution, and label the variable values.

1. 0 1 4 6 5 9 2
2. 0 1 5 4 9 6 2
3. 0 4 2 6 1 5 9
4. 9 4 0 5 1 2 6
5. 4 0 1 6 9 2 5
6. 6 4 9 5 0 1 2

Question 5. Fork Me [10 MARKS]

```
int main()
{
    int c = 0;
    for (int i = 0; i < 3; i++) {
        if (fork() == 0) {
            printf("%d␣", c + i);
            fflush(stdout);
        } else {
            c = c + i + 3;
        }
    }
    return 0;
}
```



Part (a) [2 MARKS] In the code shown above, how many times is `fork()` invoked?

7 times. In the tree diagram shown above, each circle is a fork.

Part (b) [4 MARKS] Circle all the output sequences shown below that are valid program output. Marks will be deducted for incorrect answers. Hint: Make a diagram that shows program execution, and label the variable values.

1. 0 1 4 6 5 9 2
2. 0 1 5 4 9 6 2
3. 0 4 2 6 1 5 9
4. 9 4 0 5 1 2 6
5. 4 0 1 6 9 2 5
6. 6 4 9 5 0 1 2

Note that only the child process prints values. These values are shown in green above. The only dependencies that exist are 0 -> 1 -> 2, 0 -> 6, 4 -> 5.

Question

already in Previous class

GATE CSE 2005 | Question: 72

27,646 views

98

```
Consider the following code fragment:
if (fork() == 0)
{
   a = a + 5;
   printf("%d, %p n", a, &a);
}
else
{
   a = a - 5;
   printf ("%d, %p n", a,& a);
}
```

Let u, v be the values printed by the parent process and x, y be the values printed by the child process. Which one of the following is **TRUE**?

A. $u = x + 10$ and $v = y$
B. $u = x + 10$ and $v! = y$
C. $u + 10 = x$ and $v = y$
D. $u + 10 = x$ and $v! = y$

exec System call