Process

main() {
thread 1
}

Process

$\boxed{\text{OS}}$

muti process
{ thread

$\boxed{\text{Process}}$

muti threads

main

int x=5;
fork();
x+=1;

copy

main

int x=5;
fork();

memory space

3.1 Using the program shown in Figure 3.30, explain what the output will be at `LINE A`.

<wait.h>

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int value = 5;

int main()
{
pid_t pid;

   pid = fork();

   if (pid == 0) { /* child process */
      value += 15;
      return 0;
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE A */
      return 0;
   }
}
```

NULL

neg	0	child

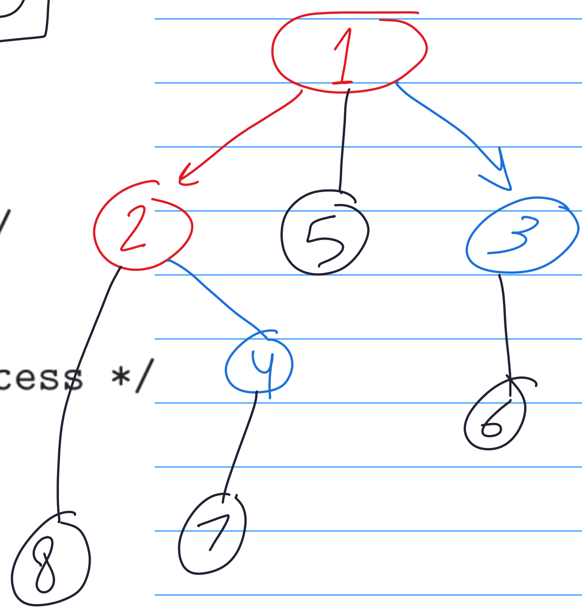
Par	Child
pid=child	pid=0;
Value=5;	~~Value=5;~~
	Value=20;

5

3.2 Including the initial parent process, how many processes are created by the program shown in Figure 3.31?

$\boxed{8}$

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

int main()
{
    /* fork a child process */
    fork();

    /* fork another child process */
    fork();

    /* and fork another */
    fork();

    return 0;
}
```



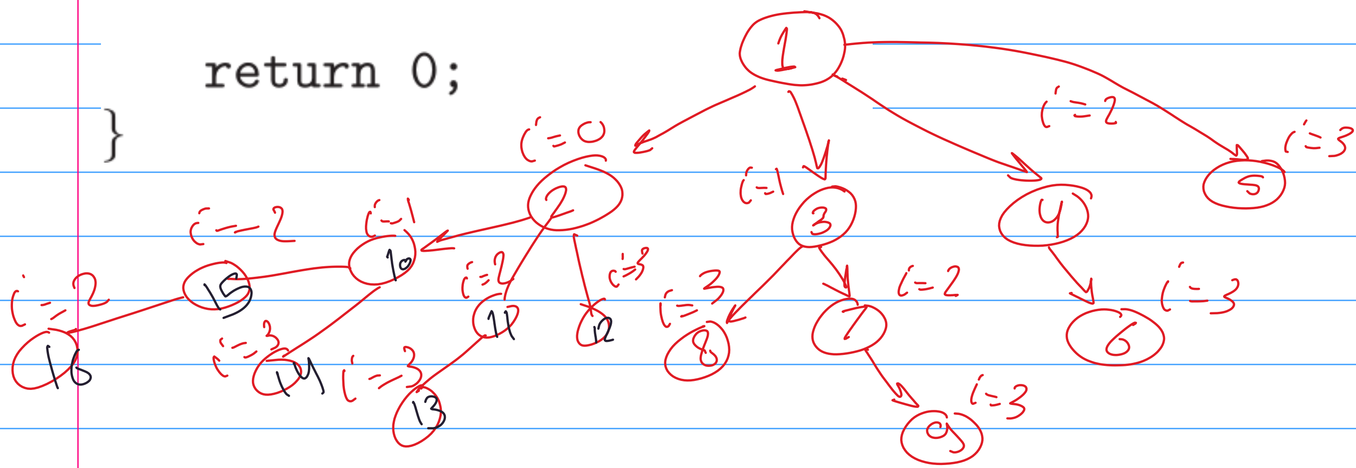
3.12 Including the initial parent process, how many processes are created by the program shown in Figure 3.32?

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

int main()
{
    int i;

    for (i = 0; i < 4; i++)
        fork();

    return 0;
}
```

$\boxed{2^4 = 16}$

Loop unrolling

```
fork();
fork();
fork();
fork();
```

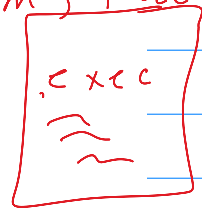


3.13 Explain the circumstances under which which the line of code marked `printf("LINE J")` in Figure 3.33 will be reached.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid;

   /* fork a child process */
   pid = fork();

   if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
   }
   else if (pid == 0) { /* child process */
      execlp("/bin/ls","ls",NULL);
      printf("LINE J");
   }
   else { /* parent process */
      /* parent will wait for the child to complete */
      wait(NULL);
      printf("Child Complete");
   }

   return 0;
}
```

exec("ls", arg)

my process



Child && exec failed

3.14 Using the program in Figure 3.34, identify the values of `pid` at lines `A`, `B`, `C`, and `D`. (Assume that the actual pids of the parent and child are 2600 and 2603, respectively.)

Par | Child
2600 | 2603

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

int main()
{
pid_t pid, pid1;

    /* fork a child process */
    pid = fork();

    if (pid < 0) { /* error occurred */
      fprintf(stderr, "Fork Failed");
      return 1;
    }
    else if (pid == 0) { /* child process */
      pid1 = getpid();
      printf("child: pid = %d",pid); /* A */  0
      printf("child: pid1 = %d",pid1); /* B */  2603
    }
    else { /* parent process */
      pid1 = getpid();  2600
      printf("parent: pid = %d",pid); /* C */  2603
      printf("parent: pid1 = %d",pid1); /* D */  2600
      wait(NULL);
    }

    return 0;
}
```

Par | Child
pid = child = 2603 | pid = 0
 | pid1 = 2603

3.17 Using the program shown in Figure 3.35, explain what the output will be at lines X and Y.

```
#include <sys/types.h>
#include <stdio.h>
#include <unistd.h>

#define SIZE 5

int nums[SIZE] = {0,1,2,3,4};

int main()
{
int i;
pid_t pid;

  pid = fork();

  if (pid == 0) {
    for (i = 0; i < SIZE; i++) {
      nums[i] *= -i;
      printf("CHILD: %d ",nums[i]); /* LINE X */
    }
  }
  else if (pid > 0) {
    wait(NULL);
    for (i = 0; i < SIZE; i++)
      printf("PARENT: %d ",nums[i]); /* LINE Y */
  }

  return 0;
}
```

Par
pid = child > 0
num {0 - - - 4}

Child
pid = 0
nums = {0 - - 4}
0, -1, -4, -9
-16

0, 1, 2, 3, 4

4.15 Consider the following code segment:

```
pid_t pid;

pid = fork();
if (pid == 0) { /* child process */
    fork();
    thread_create( . . .);
}
fork();
```

1

2

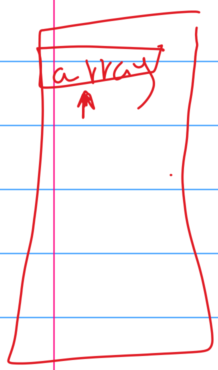
2

$+2$

$\boxed{6}$ (1 + 2) 2*

a. How many unique processes are created? 6

b. How many unique threads are created? 6 + 2 = 8

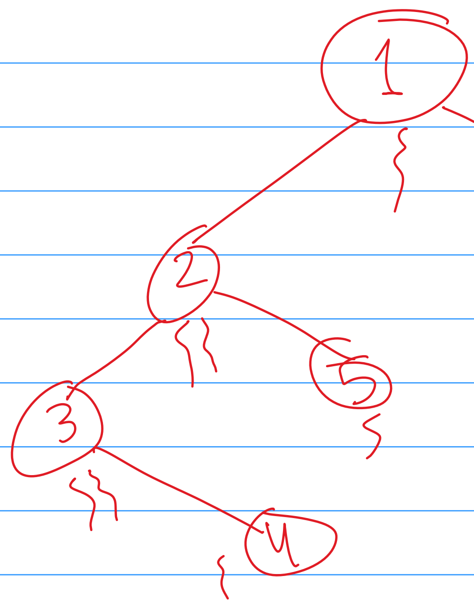


sort, max, min

thread_create(sort, -- --); life

(max

(min



6

8

4.17 The program shown in Figure 4.16 uses the Pthreads API. What would be the output from the program at LINE C and LINE P?

```c
#include <pthread.h>
#include <stdio.h>

#include <types.h>

int value = 0;
void *runner(void *param); /* the thread */

int main(int argc, char *argv[])
{
pid_t pid;
pthread_t tid;
pthread_attr_t attr;

pid = fork();

   if (pid == 0) { /* child process */
      pthread_attr_init(&attr);
      pthread_create(&tid,&attr,runner,NULL);
      pthread_join(tid,NULL);
      printf("CHILD: value = %d",value); /* LINE C */
   }
   else if (pid > 0) { /* parent process */
      wait(NULL);
      printf("PARENT: value = %d",value); /* LINE P */
   }
}

void *runner(void *param) {
   value = 5;
   pthread_exit(0);
}
```

Par

Pid >0

Value=0

Child

Pid =0

Value = 0

thread end

Value = 5

LINE C: 5

LINE P: 0