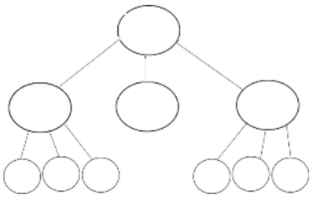
```
#include <pthread.h>
#include <iostream>
#include <iomanip>
#define NOPER 4

struct operation {
  int val1, val2, op;
   double result;
};

void *calculator(void *pos_void_ptr)
{
  struct operation *pos_ptr = (struct operation *)pos_void_ptr;
   switch(pos_ptr->op) {
      case 0: pos_ptr->result = pos_ptr->val1 + pos_ptr->val2;
            break;
      case 1: pos_ptr->result = pos_ptr->val1 - pos_ptr->val2;
            break;
      case 2: pos_ptr->result = pos_ptr->val1 * pos_ptr->val2;
            break;
      case 3: if (pos_ptr->val2 != 0)
                pos_ptr->result = (double) pos_ptr->val1 / pos_ptr->val2;
            else
                pos_ptr->result = 0;
            break;
   }
  return NULL;
}

int main()
{
  static struct operation operations[NOPER];
  pthread_t tid[NOPER];

  for(int i=0;i<NOPER;i++) {
    operations[i].op = i;
      std::cin >> operations[i].val1;
      std::cin >> operations[i].val2;
    if(pthread_create(&tid[i], NULL, calculator, &operations[i])) {
      fprintf(stderr, "Error creating thread\n");
      return 1;
    }
  }
  // Wait for the other threads to finish.
  for (int i = 0; i < NOPER; i++)
       pthread_join(tid[i], NULL);
   for (int i = 0; i < NOPER; i++) {
      switch(operations[i].op) {
         case 0: std::cout << operations[i].val1 << " + " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
               break;
         case 1: std::cout << operations[i].val1 << " - " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
               break;
         case 2: std::cout << operations[i].val1 << " * " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
               break;
         case 3: std::cout << operations[i].val1 << " / " << operations[i].val2 << " = " << std::fixed << std::setprecision(2) << operations[i].result << std::endl;
               break;
      }
   }
   return 0;
}
```

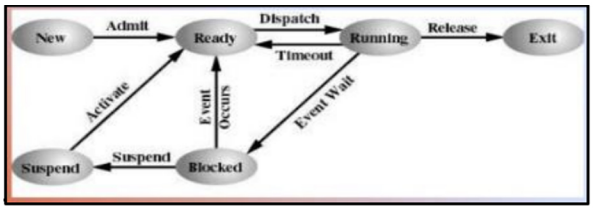


```
int main()
{
 pid_t pid;
 int i,j;
 for (i =0; i <3; i++)
 if((pid=fork())==0) // child process
 break;
 if (pid == 0)
 {
 if (i%2 == 0)
 {
 for (j =0; j <3; j++)
 if((pid=fork())==0) // child process
 break;
 }
 }
 else // parent process
 for (int i=0;i<3;i++)
 wait(NULL);
 sleep(10);
```

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

{
  pid_t pid;
  std::cout << "I am the parent process" << std::endl;
  for(int i=0;i<3;i++) {
     pid = fork();
     if (pid == 0) {
        std::cout << "I am the child process " << i << std::endl;
        if (i==1) {
           pid = fork();
           if (pid == 0) {
              std::cout << "I am a grandchild process from child process " << i << std::endl;
              _exit(0);
           }
           wait(0);
        }
        _exit(0);
     }
     wait(0);
  }
```

```
{
  for (int i = 0; i < 4; i++) {
    pid_t pid = fork();
    if (pid == 0) {
      if ((i % 2) != 0) {
        for (int j = 0; j < 4 - i; j++) {
          pid = fork();
          if (pid == 0)
            break;
        }
        if (pid != 0) {
          for (int j = 0; j < 4 - i; j++)
            wait(0);
        }
        if (pid == 0) {
          cout << "I am a grandchild process" << endl;
          _exit(0);
        }
      }
      sleep(1);
      cout << "I am a child process" << endl;
      _exit(0);
    }
  }
  for (int i = 0; i < 4; i++)
    wait(0);
  cout << "I am the parent process" << endl;
```




```
#define NTHREADS 5
#define COUNT 5
void *inc_x(void *x_void_ptr)
{
 int *x_ptr = (int *)x_void_ptr;
 int i;
 for (i=0; i< COUNT; i++)
 (*x_ptr)++;
 return NULL;
}
int main()
{
 static int x = 0;
 pthread_t tid[NTHREADS];
 int i;
 printf("Initial value of x: %d\n", x);
 for(i=0;i<NTHREADS;i++)
 {
 /*if(false) doesn't execute -> pthread_create on success returns 0 = false
 **else returns true and executes the print statements.
 **int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
 ** void *(*start_routine) (void *), void *arg); */
 if(pthread_create(&tid[i], NULL, inc_x, &x))
 {
 fprintf(stderr, "Error creating thread\n");
 return 1;
 }
 }
 // Wait for the other threads to finish - > pthread_join()
 for (i = 0; i < NTHREADS; i++)
 pthread_join(tid[i], NULL);
 printf("Final value of x: %d\n",x);
```