Processes and Threads

- Can you fill the blanks to create the child process to compute the sum of the array?

```
int sum = 0;
int array[] = {1, 2, 3, 4};

void calculate_sum() {
    for(int i = 0; i < 4; i++)
        sum += array[i];
}
int main() {
    ___________ // blank A: create child process
    if(______) { // blank B: distinguish execution paths
          _________ // blank C: compute
          printf("sum = %d\n", sum);
    } else {
        ___________// blank D: wait for child.
    }
    return 0;
}
```

- Can you fill the blanks to create a thread to compute the sum of the array?

```
int sum = 0;
int array[] = {1, 2, 3, 4};

void* calculate_sum(void* arg) {
     for(int i = 0; i < 4; i++)
          sum += array[i];
     return NULL;
}
int main() {
     pthread_t t;
     pthread_attr_t attr;
     pthread_attr_init(&attr);
     _________ // blank A: create thread
     __________// blank B: wait for thread
     printf("sum = %d\n", sum);
     return 0;
}
```

- Show possible output
  - Q1: What would be the output of the following code?
  - Q2: Based on the given code, what would be the output if line B is commented out and line C is uncommented?
  - Q3: Based on the given code, what would be the output if both line B and D are commented?
  - Q4: Based on the given code, what would be the output if line A is replaced with vfork() and line E is uncommented?

```
int counter = 0;

void* add_counter(void* arg) {
    counter = counter + 10;
    return NULL;
}

int main() {
    pthread_t t1, t2, t3;
    pid_t pid = fork(); // line A
    if (pid == 0) {
          pthread_create(&t1, NULL, add_counter, NULL);
          pthread_join(t1, NULL);  //line B
          printf("CHILD: counter = %d\n", counter);
          pthread_create(&t2, NULL, add_counter, NULL);
          // pthread_join(t1, NULL); // line C
          pthread_join(t2, NULL); // line D
          printf("CHILD: counter = %d\n", counter);
          //_exit(0); // line E
    } else {
          printf("PARENT: counter = %d\n", counter);
          wait(NULL);
          pthread_create(&t3, NULL, add_counter, NULL);
          pthread_join(t3, NULL);
          printf("PARENT: counter = %d\n", counter);
    }
    return 0;
}
```

- Q1: What would be the output for the following code?
- Q2: Based on the given code, what would be the output if line B and C are uncommented?
- Q3: Based on the given code, what would be the output if line A, B and C are uncommented?

```
void* print_message(void* arg) {
    printf("Hello from thread!”);
    return NULL;
}

int main() {

    fork();
    // fork(); // line A
    printf("Hello from process!”);
    pthread_t t1;
    // pthread_create(&t1, NULL, print_message, NULL);//line B
    // pthread_join(t1, NULL); // line C
    printf("Main process finished\n");
    return 0;
}
```

Interprocess Communication (IPC)

- Can you fill the blanks?
  - Create child process to compute the sum of the array
  - Child process share the results with parent using shared memory

```
int array[] = {1, 2, 3, 4};

void calculate_sum(______) { // blank A: setup parameter
    int s = 0;
    for(int i = 0; i < 4; i++)
        s += array[i];
    ______ // blank B: save results in shared memory
}
int main() {
    // blank C: create shared memory (following line)
    int *sum = ____(0, sizeof(int), PROT_READ | PROT_WRITE,
                    MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    ___________ // blank D: create child process
    if(______) { // blank E: distinguish execution paths
          _________ // blank F: compute
    } else {
        ___________// blank G: wait for child.
        printf("sum = %d\n",___); // blank H: print result
    }
    return 0;
}
```

- Can you fill the blanks?
  - Sender process sends a series of random integers via message queue (mq_send can only send strings, how to modify it to send integers?)
  - Receiver process receive the integers sequences and sum every three consecutive numbers and print the sum. (again, mq_receive can only receive strings, how to modify it to receive integers?)

```
#define QUEUE_NAME "/my_mq"
#define MAX_SIZE   4 //sizeof(int)
#define MSG_COUNT  9

int main() {
    struct mq_attr attr;
    attr.mq_flags = 0;          // 0 = blocking mode
    attr.mq_maxmsg = 3;         // queue can hold 3 messages
    attr.mq_msgsize = MAX_SIZE;
    attr.mq_curmsgs = 0;
    // blank A: create a message queue
    mqd_t mq = _____(QUEUE_NAME, O_CREAT | O_WRONLY, 0666, &attr);

    for (int i = 0; i < MSG_COUNT; i++) {
        int msg = rand();
        _____________________________; // blank B: send message
        sleep(1);
    }
    mq_close(mq);
    return 0;
}
```

```
#define QUEUE_NAME "/my_mq"
#define MAX_SIZE   4 //sizeof(int)
#define MSG_COUNT  9

int main() {
    // blank A: open message queue
    mqd_t mq = _____(QUEUE_NAME, O_RDONLY);

    _______ // blank B: create a buffer
    for (int i = 0; i < MSG_COUNT; i++) {
        _____________________________ // blank C: receive message
        if (____________) { // blank D: when to print?
            printf("sum = %d\n",_________________); // blank D: what to print
        }
    }
    mq_close(mq);
    return 0;
}
```

- Show possible output
  - Q1: What might be the output of the following code?
  - Q2: What might be the output if line A is uncommented and line B and commented out?

```
int main() {
    int *counter = mmap(0, sizeof(int), PROT_READ | PROT_WRITE,
                        MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    *counter = 0;

    if (fork() == 0) {
        for (int i = 0; i < 10; i++)
            (*counter)++;
    } else {
        // wait(NULL)   // line A
        for (int i = 0; i < 10; i++)
            (*counter)++;
        wait(NULL);     // line B
        printf("Final counter = %d\n", *counter);
    }
}
```

- Show possible output
    - Q1: Will the final sum be correct? Why or why not?
    - Q2: Which parts of memory are shared across both processes?
    - Q3: How could this be fixed so that the result is accurate and deterministic?

```
int *sum;

void *add_half(void *arg) {
    int start = ((int*)arg)[0];
    int end = ((int*)arg)[1];
    for (int i = start; i <= end; i++)
        *sum += i;
    return NULL;
}

int main() {
    sum = mmap(0, sizeof(int), PROT_READ | PROT_WRITE,
               MAP_SHARED | MAP_ANONYMOUS, -1, 0);
    *sum = 0;

    if (fork() == 0) {
        int range[] = {1, 50};
        pthread_t tid;
        pthread_create(&tid, NULL, add_half, range);
        pthread_join(tid, NULL);
        _exit(0);
    } else {
        int range[] = {51, 100};
        pthread_t tid;
        pthread_create(&tid, NULL, add_half, range);
        pthread_join(tid, NULL);
        wait(NULL);
        printf("Final sum = %d\n", *sum);
    }
}
```