

Total: 4pts; 1pt for each question.

1. Given that you can create multiple threads to perform different tasks within a program, explain why you might still need to use fork.
2. The code snippet below needs to be fixed. Point out the problem. (Assume ... does not do anything wrong).

```
int main(void) {
    signal(SIGALRM, sig_alm);
    alarm(60);
    if (setjmp(env_alm) != 0) {
        /* handle timeout */
        ...
    }
    ...
}

static void sig_alm(int signo) {
    longjmp(env_alm, 1);
}
```

3. Consider the following code snippet for a bounded queue. Consider multiple POSIX threads that could each execute the enqueue() and dequeue() functions. Fill out the missing parts (A), (B), (C), and (D) to achieve proper synchronization with conditional variable and mutex. (0.5pts). Next, explain why the “while (q->count...” checks in the two functions are crucial.

```
typedef struct {
    int data[MAX_SIZE];
    int front;
    int rear;
    int count;
    pthread_mutex_t mutex;
    pthread_cond_t not_full;
    pthread_cond_t not_empty;
} Queue;

void enqueue(Queue *q, int item) {
```

```

pthread_mutex_lock(&q->mutex);

while (q->count == MAX_SIZE) {
    _____(A)_____; // Wait when queue is full
}

q->data[q->rear] = item;
q->rear = (q->rear + 1) % MAX_SIZE;
q->count++;

_____(B)_____; // Signal that queue is not empty
pthread_mutex_unlock(&q->mutex);
}

int dequeue(Queue *q) {
    pthread_mutex_lock(&q->mutex);

    while (q->count == 0) {
        _____(C)_____; // Wait when queue is empty
    }

    int item = q->data[q->front];
    q->front = (q->front + 1) % MAX_SIZE;
    q->count--;

    _____(D)_____; // Signal that queue is not full
    pthread_mutex_unlock(&q->mutex);
    return item;
}

```

4. Using `mmap()` to perform IPC in a Unix environment is possible. Explain how you would do that with a code snippet. (incomplete code is OK; i.e., the code does not have to be compiled)