期中复习

操作系统期中考试C++伪代码编程题预测与复习

根据你的考试范围(进程线程，不包括死锁)和题型(3道C++伪代码编程题)，我为你总结了可能考察的所有编程题类型及对应的复习内容：

一、基础进程/线程创建题 (必考1题)

1. 多进程创建(fork)

```
// 创建多个子进程执行任务
#include <unistd.h>
#include <sys/wait.h>

int main() {
    for(int i=0; i<3; i++) {
        pid_t pid = fork();
        if(pid == 0) { // 子进程
            printf("子进程%d, PID=%d\n", i, getpid());
            _exit(0); // 子进程退出
        }
    }

    // 父进程等待所有子进程
    while(wait(NULL) > 0);
    printf("所有子进程完成\n");
    return 0;
}
```

2. 多线程创建(pthread)

```
// 创建多个线程执行不同任务
#include <pthread.h>

void* task(void* arg) {
    int id = *(int*)arg;
    printf("线程%d正在工作\n", id);
    return NULL;
}

int main() {
    pthread_t threads[3];
    int ids[3] = {1,2,3};
```

```
    for(int i=0; i<3; i++) {
        pthread_create(&threads[i], NULL, task, &ids[i]);
    }

    for(int i=0; i<3; i++) {
        pthread_join(threads[i], NULL);
    }

    printf("所有线程完成\n");
    return 0;
}
```

二、线程同步题 (必考1-2题)

1. 互斥锁基础应用

```
// 多线程安全计数器
#include <pthread.h>

int counter = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;

void* increment(void* arg) {
    for(int i=0; i<10000; i++) {
        pthread_mutex_lock(&mutex);
        counter++;
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    pthread_t t1, t2;
    pthread_create(&t1, NULL, increment, NULL);
    pthread_create(&t2, NULL, increment, NULL);

    pthread_join(t1, NULL);
    pthread_join(t2, NULL);

    printf("最终计数器值: %d\n", counter); // 应该是20000
    return 0;
}
```

2. 条件变量实现生产者-消费者

```cpp
// 有限缓冲区生产者-消费者
#include <pthread.h>
#include <queue>

#define MAX 10
std::queue<int> buffer;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t cond_full = PTHREAD_COND_INITIALIZER;
pthread_cond_t cond_empty = PTHREAD_COND_INITIALIZER;

void* producer(void* arg) {
    for(int i=0; i<20; i++) {
        pthread_mutex_lock(&mutex);
        while(buffer.size() >= MAX) {
            pthread_cond_wait(&cond_full, &mutex);
        }
        buffer.push(i);
        printf("生产: %d\n", i);
        pthread_cond_signal(&cond_empty);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

void* consumer(void* arg) {
    for(int i=0; i<20; i++) {
        pthread_mutex_lock(&mutex);
        while(buffer.empty()) {
            pthread_cond_wait(&cond_empty, &mutex);
        }
        int item = buffer.front();
        buffer.pop();
        printf("消费: %d\n", item);
        pthread_cond_signal(&cond_full);
        pthread_mutex_unlock(&mutex);
    }
    return NULL;
}

int main() {
    pthread_t prod, cons;
    pthread_create(&prod, NULL, producer, NULL);
    pthread_create(&cons, NULL, consumer, NULL);

    pthread_join(prod, NULL);
    pthread_join(cons, NULL);
```

```
    return 0;
}
```

3. 读写锁实现读者-写者问题

```
// 读者优先的读写锁实现
#include <pthread.h>

int shared_data = 0;
int readers_count = 0;
pthread_mutex_t mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t write_mutex = PTHREAD_MUTEX_INITIALIZER;

void* reader(void* arg) {
    pthread_mutex_lock(&mutex);
    readers_count++;
    if(readers_count == 1) {
        pthread_mutex_lock(&write_mutex);
    }
    pthread_mutex_unlock(&mutex);

    printf("读取数据: %d\n", shared_data);

    pthread_mutex_lock(&mutex);
    readers_count--;
    if(readers_count == 0) {
        pthread_mutex_unlock(&write_mutex);
    }
    pthread_mutex_unlock(&mutex);

    return NULL;
}

void* writer(void* arg) {
    pthread_mutex_lock(&write_mutex);
    shared_data++;
    printf("写入数据: %d\n", shared_data);
    pthread_mutex_unlock(&write_mutex);
    return NULL;
}
```

三、进程间通信题 (可能考1题)

1. 管道通信

```
// 父子进程通过管道通信
#include <unistd.h>
```

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

int main() {
    int fd[2];
    pipe(fd);

    if(fork() == 0) { // 子进程
        close(fd[0]); // 关闭读端
        char msg[] = "Hello from child";
        write(fd[1], msg, sizeof(msg));
        close(fd[1]);
        exit(0);
    } else { // 父进程
        close(fd[1]); // 关闭写端
        char buf[100];
        read(fd[0], buf, sizeof(buf));
        printf("父进程收到: %s\n", buf);
        close(fd[0]);
        wait(NULL);
    }
    return 0;
}
```

2. 共享内存

```c
// 使用共享内存进行进程通信
#include <sys/ipc.h>
#include <sys/shm.h>
#include <unistd.h>

int main() {
    int shmid = shmget(IPC_PRIVATE, 1024, 0666|IPC_CREAT);
    char* shm = (char*)shmat(shmid, NULL, 0);

    if(fork() == 0) { // 子进程
        sprintf(shm, "共享内存消息");
        shmdt(shm);
        exit(0);
    } else { // 父进程
        wait(NULL);
        printf("父进程读取: %s\n", shm);
        shmdt(shm);
        shmctl(shmid, IPC_RMID, NULL);
    }
    return 0;
}
```

四、综合应用题 (可能考1题)

1. 线程池简单实现

```
// 简易线程池实现
#include <pthread.h>
#include <queue>

#define THREAD_NUM 4

struct Task {
    void (*func)(void*);
    void* arg;
};

std::queue<Task> task_queue;
pthread_mutex_t queue_mutex = PTHREAD_MUTEX_INITIALIZER;
pthread_cond_t queue_cond = PTHREAD_COND_INITIALIZER;

void* worker_thread(void* arg) {
    while(true) {
        pthread_mutex_lock(&queue_mutex);
        while(task_queue.empty()) {
            pthread_cond_wait(&queue_cond, &queue_mutex);
        }
        Task task = task_queue.front();
        task_queue.pop();
        pthread_mutex_unlock(&queue_mutex);

        task.func(task.arg);
    }
    return NULL;
}

void init_thread_pool(pthread_t* threads) {
    for(int i=0; i<THREAD_NUM; i++) {
        pthread_create(&threads[i], NULL, worker_thread, NULL);
    }
}

void add_task(void (*func)(void*), void* arg) {
    pthread_mutex_lock(&queue_mutex);
    task_queue.push({func, arg});
    pthread_cond_signal(&queue_cond);
    pthread_mutex_unlock(&queue_mutex);
}
```

复习建议

1. **重点掌握**：多线程同步(互斥锁、条件变量)、生产者-消费者模型、进程创建与通信

2. **理解原理**：每个同步机制为什么需要、解决了什么问题
3. **伪代码要点**：
    - 函数签名和关键系统调用要写对
    - 同步机制的使用顺序不能错
    - 资源释放不能遗漏
4. **典型错误**：
    - 忘记解锁互斥锁
    - 条件变量使用不当(没有while循环检查条件)
    - 没有正确处理进程/线程退出

考试时注意：

- 伪代码要清晰表达算法逻辑
- 关键系统调用名称要写准确
- 同步机制的使用要完整(初始化、加锁/解锁、销毁等)

祝你考试顺利！如果需要更详细解释某个例子或概念，可以随时告诉我。