1. (8pts.) Using only system calls such as read, write, fork, exec, dup, pipe, write a program fragment which will open a file named “outfile” and cause writes to stdout to be written to outfile.

   fd = open("outfile", O_CREAT);
   close(stdout);
   dup(fd);

2. (8pts.) Using only system calls such as read, write, fork, exec, dup, pipe, write a program fragment which creates a new process which will execute the program “du”.

   if (fork()==0) { /* child */
      execl("du", NULL);
   }

3. (10pts.) Using only system calls such as read, write, fork, exec, dup, pipe, write a program fragment which creates a process and which sends a string “this is it” from the creating process to the created process via a pipe.

   int fd[2];
   pipe(fd);
   if (fork()==0) { /* child */
      int size = strlen("this is it");
      read(fd[0], buffer, size);
   } else { /* parent */
      write(fd[1], "this is it", size);
   }
4. (8 pts.) Consider two programs \( p_0 \) and \( p_1 \) where \( p_0 \) has labels A and C, and \( p_1 \) has labels B and D. Use semaphores (and show their initialization) which will ensure that the code executes in order A,B,C, and D even though \( p_0 \) and \( p_1 \) are different processes.

\[
\begin{array}{c|c}
 p_0 & p_1 \\
 A & \text{wait(sem1)} \\
 \text{signal(sem1)} & B \\
 \text{wait(sem0)} & \text{signal(sem0)} \\
 C & \text{wait(sem1)} \\
 \text{signal(sem1)} & D \\
\end{array}
\]

5. (12pts.) Consider a system with \( N_1 \) units of resource 1 and \( N_2 \) units of resource 2. Write two procedures which are executed concurrently:

- allocate(n1, n2) allocate n1 elements of resource 1 and n2 elements of resource 2.
- release(n1, n2) return the resources.

```cpp
bool allocate(int n1, int n2) {
    wait(mutex);
    if ((n1 <= avail1) && (n2 <= avail2)) {
        avail1 -= n1;
        avail2 -= n2;
        signal(mutex);
        return true;
    }
    signal(mutex);
    return false;
}

release(int n1, int n2) {
    wait(mutex);
    avail1 += n1;
    avail2 += n2;
    signal(mutex);
}
```

6. (8pts.) What are the 4 necessary and sufficient conditions for deadlock?

\textit{ans. The terms and their meaning should be given.}

(a) mutual exclusion
(b) hold and wait
(c) no preemption
(d) circular wait

7. (6pts.) What is the relationship between the wait-for graph and deadlock?
ans. a cycle in the wait-for graph which has only a single copy of each resources means the system is in deadlock.

8. (12pts.) What happens on a system call when there are base and limits registers?

(a) base=0, limit=MEMSIZE
(b) set the privilege bit
(c) enter through the trap vector

9. (12pts.) What are the possible outcomes of interleavings, assuming all variables are shared and initially 0 in the below questions. Show the value of the variables at completion and an instruction interleaving which achieves each set of values.

• Each of two processes execute the sequence
  \[ i += 1; \]
  \[ i += 1; \]
Which is compiled into 6 machine language instruction (load-increment-store-load-increment-store)?
ans. 2 \( (a1,b1,a2,b2,a3,b3,a4,b4,a5,b5,a6,b6) \), 3 \( (a1,b1,a2,b3,a3,b3,a4,a5,a6,b4,b5,b6) \),
4 \( (a1,a2,a3,a4,a5,a6,b1,b2,b3,b4,b5,b6) \)

• One process executes \( i += 2 \) and the other process \( j += 1 \)?
  ans. always \( i = 2 \) and \( j = 1 \) (no interference)

10. (10pts.) What are the architectural requirements to implement independent processes on a single processor?

   (a) memory management such as base and limits,
   (b) privileged state and privileged instructions
   (c) trap instruction
   (d) timer interrupt

11. (6pts.) Give an example of a sequence of code which interferes with itself (ie if run in two separate processes) but which needs no synchronization code to operate atomically.
   \[ i = 5; \]