**1. (10 pts.) Unix writes.** Write the `tac` program, which reads from standard input and writes to a file which is its first argument, that is it is invoked ‘% tac outfile’.

```bash

```
2. (15 pts.) Unix Semaphores and Shared Memory Write a unix program fragment which will allocate a semaphore and shared memory to contain an array of N integers (initially 0) pointed to by p. Then write an atomic update function `atomicUpdate(int i)`, which regardless of the number of processes sharing, will atomically increment $p[i]$ and return the value of $p[i]$ before increment.
3. **(10 pts.) Synchronization** Write two procedures using semaphores for synchronization that use shared array, $a$ of size $N$:

(a) `void addTo(int i, int x);` atomically adds $x$ to $a[i]$.

(b) `int sum();` atomically return $\sum_{i=0}^{N-1} a[i]$
4. (15 pts.) Deadlock

(a) What does the following wait-for graph tell you about whether there is deadlock and what processes are involved?

(b) Consider programs which request resources A and B. How would you avoid deadlock by allowing preemption?

(c) Consider programs which request resources one at a time always in the order A, C, B. Can this system deadlock? Explain.

(d) How would you use the Banker’s algorithm to detect deadlock?
5. **(5 pts.) Paging.** What are the advantages of paging over base-and-limits registers?

6. **(10 pts.) Monitors.** Write a monitor which maintains a stack of $N$ elements and has two public methods, push and pop. The method, `void push(int x)` inserts an element in the stack, entering the wait state if the stack is full. The method, `int pop()` removes the top element from the stack and returns its value, and enters the wait state if the stack is empty.
7. (10 pts.) Bankers Algorithm. Consider the following state of the Banker’s Algorithm.
Is it safe to allocate 1 unit of $R_1$ to $P_3$? Show all work.

<table>
<thead>
<tr>
<th></th>
<th>$R_1$</th>
<th>$R_2$</th>
<th>$R_4$</th>
<th>finished</th>
</tr>
</thead>
<tbody>
<tr>
<td>$P_1$</td>
<td>0/2</td>
<td>1/2</td>
<td>0/0</td>
<td>false</td>
</tr>
<tr>
<td>$P_2$</td>
<td>0/0</td>
<td>0/1</td>
<td>1/1</td>
<td>false</td>
</tr>
<tr>
<td>$P_3$</td>
<td>1/1</td>
<td>1/1</td>
<td>1/2</td>
<td>false</td>
</tr>
<tr>
<td>total resources</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>available resources</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
8. (10 pts.) Processes.

(a) What is finite progress, and how is it ensured in an operating system?

(b) What mechanisms are used to prevent one process from attacking another process?

(c) What tasks must be done by the operating system kernel for the process abstraction?
9. **(8pts.) Two-phase locking.**

   (a) List the steps (including conditions) necessary to obtain a read lock.

   (b) List the steps (including conditions) necessary to obtain a write lock on a location which is not currently locked.

   (c) List the steps (including conditions) necessary to upgrade a lock on a location for which the upgrade can be performed.

10. **(7pts.) Scheduling.**

    (a) When and from where does a process get moved to the ready state?

    (b) When and from where does a process get moved to the run state?