CSCI-3753: Operating Systems Fall 2019

Abigail Fernandes

Department of Computer Science

University of Colorado Boulder

A little something to keep you motivated!

Thank you for registering to attend the Apple Networking Day.

You are confirmed to attend the event tomorrow, September 24th, from **9:00am** in Idea Forge Commons.

Below you will find a list of the skill sets we are recruiting for. **Please remember to bring multiple copies of your resume, as well as your Student ID.**

- AI, Machine Learning, NLP
- iOS/MacOS App-dev, UI/UX, Frameworks (Objective-C, C++, Swift)
- Web-dev, Front-end, Back-end
- Core OS, Embedded Systems, Kernel Programming, Drivers, GPU
- Systems Integration, Firmware, Micro-controllers, Automation
- EE, Board Design, Chip Architecture
- Digital Design
- Robotics, Mechatronics
- Product Design, Mechanical Design

Interview Grading Prep

- Grading: **50 % Code, 50 % Interview**
- DO NOT USE GLOBAL VARIABLES
- Build with “-Wall” and “-Wextra” (Resolve all warnings)
- Valgrind -> Free all allocated memory, no additional warnings and errors
- Things to submit:
  - MAKEFILE
  - README
  - Performance.txt
  - multi-lookup.c
  - multi-lookup.h
- Please prepare for the interview. (For example, Why you used a particular function, pthread related functions and their purpose)

Week 9
> Bankers Algorithm

Deadlocks

- Deadlocks exist amongst a set of processes if:
  - Every process is waiting for an event
  - This event can only be caused by another process in the set
    - Event can either be the acquisition or release of a resource

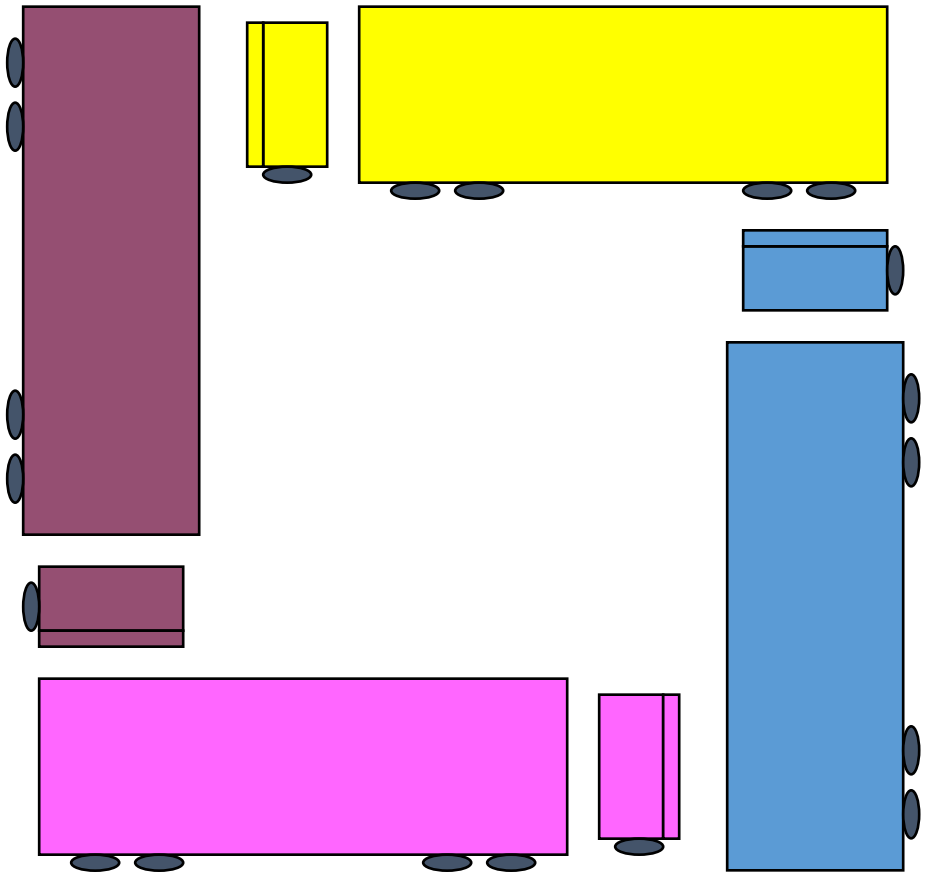
Real world deadlocks?

Truck A has to wait for truck B to move

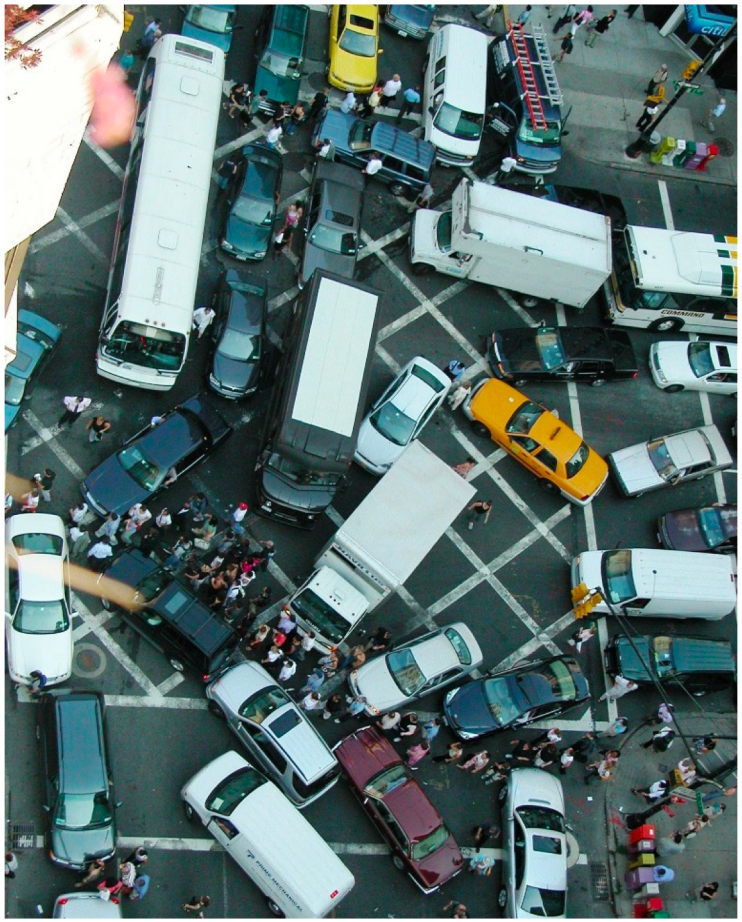
Not deadlocked

Real world deadlocks?

GRIDLOCK



Thinking of moving to NYC?



Deadlock avoidance strategy

Deadlock Avoidance

- If we have future information
  - Max resource requirement of each process before they execute
- Can we guarantee that deadlocks will never occur?
- Avoidance Approach:
  - Before granting resource, check if state is **safe**
  - If the state is safe $\Rightarrow$ no deadlock!

Safe State

- A state is said to be **safe**, if it has a process sequence

  $\{P_1, P_2, \ldots, P_n\}$, such that for each P_i,

  the resources that P_i can still request can be satisfied by the currently available resources plus the resources held by all P_j, where $j < i$

- State is safe because OS can definitely avoid deadlock
  - by blocking any new requests until safe order is executed

- This avoids circular wait condition
  - Process waits until safe state is guaranteed

Safe State Example

- Suppose there are 12 tape drives

	max need	current usage	could ask for
p0	10	5	5
p1	4	2	2
p2	9	2	7

3 drives remain

- Current state is safe because a safe sequence exists: <p1,p0,p2>
  - p1 can complete with current resources
  - p0 can complete with current+p1
  - p2 can complete with current +p1+p0

Banker's Algorithm- Introduction

- Banker's Algorithm also referred to as **Deadlock avoidance algorithm**
- Developed by **Dijkstra** (Remember shortest path in a weighted graph)

Banker's Algorithm - Introduction

- Suppose we know the "worst case" resource needs of processes in advance
  - A bit like knowing the credit limit on your credit cards. (This is why they call it the Banker's Algorithm)
- Observation: Suppose we just give some process **ALL** the resources it could need…
  - Then it will execute to completion.
  - After which it will give back the resources.
- Like a bank: If Visa just hands you all the money your credit lines permit, at the end of the month, you'll pay your entire bill, right?

Banker's Algorithm

- So…
  - A process pre-declares its worst-case needs
  - Then it asks for what it "really" needs, a little at a time
  - The algorithm decides when to grant requests
- It delays a request unless:
  - It can find a sequence of processes…
    - …. such that it could grant their outstanding need…
    - … so they would terminate…
    - … letting it collect their resources…
    - … and in this way it can execute everything to completion!

Banker's Algorithm – Data Structures

```
n: integer                    # of processes
m: integer                    # of resources

available[1..m] - available[i] is # of available
resources of type i

max[1..n, 1..m]   - max demand of each Pi for each
Ri. Max[i][j] = k means that Pi can request at
most k instances of Ri
```

Banker's Algorithm – Data Structures

allocation[1..n, 1..m] – allocation[i][j] is the current allocation of resource Rj to Pi

need[1..n, 1..m] - max # resource Rj that Pi may still request
need[i,j] = max[i,j] – allocation[i, j]

let request[i] be vector of # of resource Rj Process Pi wants

Banker's Algorithm – Data Structures

Banker's Algorithm Terminology

- Let X and Y be two vectors (think 1D - arrays)
- Then we say X ≤ Y if and only if X[i] ≤ Y[i] for all i.
- Example:

$$V1 = \begin{bmatrix} 1 \\ 7 \\ 3 \\ 2 \end{bmatrix} \quad V2 = \begin{bmatrix} 0 \\ 3 \\ 2 \\ 1 \end{bmatrix} \quad V3 = \begin{bmatrix} 0 \\ 10 \\ 2 \\ 1 \end{bmatrix}$$

then V2 ≤ V1, but V3 ≰ V1, i.e. V3 is not less than or equal to V1

Basic Algorithm

1. If request[i] > need[i] then
   error (asked for too much)
2. If request[i] > available[i] then
   wait (can't supply it now)
3. Resources are available to satisfy the request

   Let's assume that we satisfy the request. Then we would have:

   available = available - request[i]

   allocation[i] = allocation [i] + request[i]

   need[i] = need [i] - request [i]

   Now, check if this would leave us in a safe state:

   if yes, grant the request,

   if no, then leave the state as is and cause process to wait.

Safety Check

```
work[1..m] = available /* how many resources are available */

finish[1..n] = false (for all i)  /* none finished yet */

Step 1: Find an i such that finish[i]=false and
need[i] <= work
       /* find a proc that can complete its request now */
      if no such i exists, go to step 3   /* we're done */

Step 2: Found an i:
     finish [i] = true  /* done with this process */
     work = work + allocation [i]
      /* assume this process were to finish, and its
         allocation goes back to the available list */
     go to step 1

Step 3: If finish[i] = true for all i, the system is safe.
        Else Not
```

Your turn now!
Solve an Example

Banker's Algorithm Example

Considering a system with five processes **P_0 through P_4** and three **resources of type A, B, C**. Resource type A has 10 instances, B has 5 instances and type C has 7 instances. Suppose at time t_0 following snapshot of the system has been taken:

Process	Allocation	Max	Available
	A B C	A B C	A B C
P_0	0 1 0	7 5 3	3 3 2
P_1	2 0 0	3 2 2	
P_2	3 0 2	9 0 2	
P_3	2 1 1	2 2 2	
P_4	0 0 2	4 3 3	

Banker's Algorithm Example

Question1. What will be the content of the Need matrix?

Recall: Need [i, j] = Max [i, j] – Allocation [i, j]

Process	Need		
	A	B	C
P_0	7	4	3
P_1	1	2	2
P_2	6	0	0
P_3	0	1	1
P_4	4	3	1

Banker's Algorithm Example

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Banker's Algorithm Example

m=3, n=5 — Step 1 of Safety Algo

Work = Available

Work = | 3 | 3 | 2 |

	0	1	2	3	4
Finish =	false	false	false	false	false

For i = 0 — Step 2

$Need_0$ = 7, 4, 3 (7,4,3 vs 3,3,2)

Finish [0] is false and $Need_0$ > Work

So P_0 must wait — But Need ≤ Work

For i = 1 — Step 2

$Need_1$ = 1, 2, 2 (1,2,2 vs 3,3,2)

Finish [1] is false and $Need_1$ < Work

Step 3

3, 3, 2 + 2, 0, 0

Work = Work + $Allocation_1$

	A	B	C
Work =	5	3	2

	0	1	2	3	4
Finish =	false	true	false	false	false

For i = 2 — Step 2

$Need_2$ = 6 , 0, 0 (6, 0, 0 vs 5,3, 2)

Finish [2] is false and $Need_2$ > Work

So P_2 must wait

For i=3 — Step 2

$Need_3$ = 0, 1, 1 (0, 1, 1 vs 5, 3, 2)

Finish [3] = false and $Need_3$ < Work

So P_3 must be kept in safe sequence

Step 3

5, 3, 2 + 2, 1, 1

Work = Work + $Allocation_3$

	A	B	C
Work =	7	4	3

	0	1	2	3	4
Finish =	false	true	false	true	false

For i = 4 — Step 2

$Need_4$ = 4, 3, 1 (4, 3, 1 vs 7, 4, 3)

Finish [4] = false and $Need_4$ < Work

So P_4 must be kept in safe sequence

Step 3

7, 4, 3 + 0, 0, 2

Work = Work + $Allocation_4$

	A	B	C
Work =	7	4	5

	0	1	2	3	4
Finish =	false	true	false	true	true

For i = 0 — Step 2

$Need_0$ = 7, 4, 3 (7, 4, 3 vs 7, 4, 5)

Finish [0] is false and Need < Work

So P_0 must be kept in safe sequence

Step 3

7, 4, 5 + 0, 1 , 0

Work = Work + $Allocation_0$

	A	B	C
Work =	7	5	5

	0	1	2	3	4
Finish =	true	true	false	true	true

For i = 2 — Step 2

$Need_2$ = 6 , 0, 0 (6, 0, 0 vs 7, 5, 5)

Finish [2] is false and $Need_2$ < Work

So P_2 must be kept in safe sequence

Step 3

7, 5, 5 + 3, 0, 2

Work = Work + $Allocation_2$

	A	B	C
Work =	10	5	7

	0	1	2	3	4
Finish =	true	true	true	true	true

Finish [i] = true for 0 ≤ i ≤ n — Step 4

Hence the system is in Safe state

Banker's Algorithm Example

Question2. Is the system in a safe state? If Yes, then what is the safe sequence?

Yes, <*P1, P3, P4, P0, P2*> is a safe sequence

Week 9 – Checklist

- ❑ Discuss Banker's algorithm
- ❑ Get a high score on PA3 (Grading checklist)