15-213/513/613: Final Exam Review

Kashish & Ishita

Final Exam Logistics

- Cheat sheets - 2 double sided 8.5 x 11 in.
- Join Zoom, turn on video AND microphone.
- Show ID and eo.
- You will recei to the zoom call, the exam time, and more de tics soon.

- 8 categories of questions:
  - Malloc, VM, Processes, Signals, IO, Threads, ThreadSync, Multiple Choice (pre-midterm)

Virtual Memory

Virtual Memory

Virtual Address - 18 Bits

Physical Address - 12 Bits

Page Size - 512 Bytes

TLB is 8-way set associative

Cache is 2-way set associative

TLB			
Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

2-way Set Associative Cache												
Index	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3	Tag	Valid	Byte 0	Byte 1	Byte 2	Byte 3
0	7A	1	09	EE	12	64	00	0	99	04	03	48
1	02	0	60	17	18	19	7F	1	FF	BC	0B	37
2	55	1	30	EB	C2	0D	0B	0	8F	E2	05	BD
3	07	1	03	04	05	06	5D	1	7A	08	03	22

Final S-02 (#5)

Lecture 18: VM - Systems

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Ind

(D) *TLBT:* TLB Ta

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset - Location in the page

Page Size = 512 Bytes = 2^9 → Need 9 bits

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number - Everything Else

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Ind

Cache

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Ind

Cache

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A
								TLBI									

Virtual Memory

Label the following:

(A) *VPO:* Virtual Page Offset

(B) *VPN:* Virtual Page Number

(C) *TLBI:* TLB Ind

(D) TLBT: TLB Ta

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	B	B	B	B	B	B	A	A	A	A	A	A	A	A	A

TLBT TLBI

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset

(B) *PPN:* Physical Page Number

(C) *CO:* Cache Of

(D) *CI:* Cache Ind

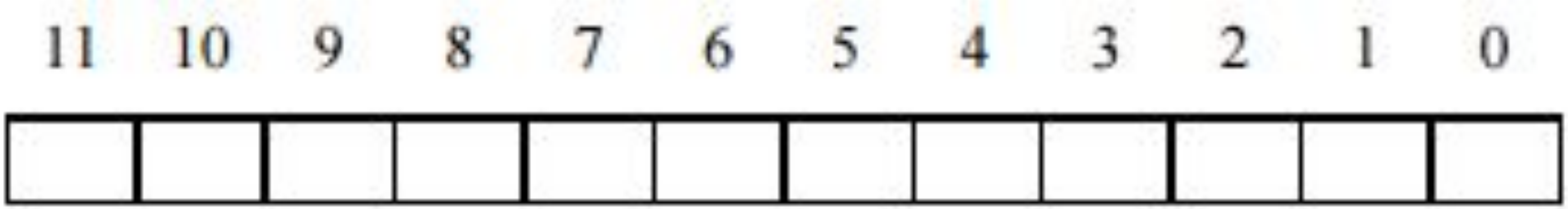
(E) *CT:* Cache Tag

11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset



Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

11	10	9	8	7	6	5	4	3	2	1	0
			A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Of

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Of

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

CO

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Of

(D) *CI:* Cache Ind

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

CO

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Of

(D) *CI:* Cache Ind

4 indices →

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

CI CO

Virtual Memory

Label the following:

(A) *PPO:* Physical Page Offset - Same as VPO

(B) *PPN:* Physical Page Number - Everything Else

(C) *CO:* Cache Of

(D) *CI:* Cache Ind

(E) *CT:* Cache Tag - Everything

11	10	9	8	7	6	5	4	3	2	1	0
B	B	B	A	A	A	A	A	A	A	A	A

Cache Tag (bits 11–4), CI (bits 3–2), CO (bits 1–0)

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation

   1 = 0001 A = 1 4 = 0100

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x??

TLB Hit: Y/N? Page Fault: : 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x??

TLB Hit: Y/N? Page Fault: : 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x??

TLB Hit: Y/N? Page Fault: : 0x??

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x6A

TLB Hit: Y/N? Page Fault: : 0x??

TLB

Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x6A

TLB Hit: Y! Page Fault: Y x??

TLB			
Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x6A

TLB Hit: Y! Page Fault: ??

TLB			
Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

TLB			
Index	Tag	PPN	Valid
0	55	6	0
	48	F	1
	00	A	0
	32	9	1
	6A	3	1
	56	1	0
	60	4	1
	78	9	0
1	71	5	1
	31	A	1
	53	F	0
	87	8	0
	51	D	0
	39	E	1
	43	B	0
	73	2	1

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio

VPN: 0 BT: 0x6A

TLB Hit: Y! Page Fault: 3

17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	0	1	0	1	0	0	1	1	1	1	1	0	1	0	0

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio
3. Put it all together:

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1									

Virtual Memory

Now to the actual question!

Q) **Translate the following address: 0x1A9F4**

1. Write down bit representation
2. Extract Informatio
3. Put it all together:

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

CO: 0x?? CI: 0x?? /N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information

CO: 0x00 CI: 0x?? /N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information

CO: 0x00 CI: 0x01 /N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information
2. Go to Cache Table

CO: 0x00 CI: 0x01 /N? Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information
2. Go to Cache Table

CO: 0x00 CI: 0x01 Value:0x??

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Q) **What is the value of the address?**

1. Extract more information
2. Go to Cache Table

CO: 0x00 CI: 0x01 Value:0xFF

11	10	9	8	7	6	5	4	3	2	1	0
0	1	1	1	1	1	1	1	0	1	0	0

Virtual Memory

Threads

Threads

Given this code, what variables do you think are shared?

Threads (Contd.)

Which variables can be shared by multiple threads simultaneously in this program?

(A) i
(B) balance
(C) instance
(D) cnt
(E) None of the above

Threads (Contd.)

Which variables can be shared by multiple threads simultaneously in this program?

(A) i
(B) balance
(C) instance
(D) cnt
(E) None of the above

Answer: B

Threads (Contd.)

(A) i is a local variable so it isn't shared.

(B) balance is a global variable so it's shared.

(C) instance is n't shared.

(D) cnt is a static variable, so it retains its value even outside the scope in which it was defined, so it isn't shared.

Threads (Contd.)

Given the withdraw() and deposit() functions, what are the possible outputs? (balance = 10 initially)


```
int deposit(int amt) {
    balance = balance + amt;
    sleep(2);
    return 0;
}
```

Threads (Contd.)

What can be the value of balance?

(A) balance: 0
(B) balance: -3
(C) balance: 1
(D) balance: 6
(E) balance: 17
(F) balance: 4

Threads (Contd.)

What can be printed at the indicated line?

(A) balance: 0
(B) balance: -3
(C) balance: 1
(D) balance: 6
(E) balance: 17
(F) balance: 4

Answer: ABDF

Threads (Contd.)

The following is one interleaving that leads to output 0:

- Thread A executes deposit(4), balance = 14
- Thread B e nce = 8

- Thread B e nce = 11
- Thread A executes withdr lance = 0

- Thread B executes withdraw(7), balance = 0

Threads (Contd.)

The following is one interleaving that leads to output -3:

- Thread A executes deposit(4), balance = 14
- Thread A s w(11) and enters the i
- Thread B executes withdr nce = 8
- Thread A computes RHS for withdraw(11) = -3
- Thread B executes deposit(3), balance = 11
- Thread A completes withdraw(11), balance = -3
- Thread B executes withdraw(7), balance = -3

Threads (Contd.)

The following is one interleaving that leads to output 6:

- Thread A executes deposit(4), balance = 14
- Thread A e balance = 3

nce = 3

- Thread B e

- Thread B executes depos ce = 6
- Thread B executes withdraw(7), balance = 6

Threads (Contd.)

The following is one interleaving that leads to output 4:

- Thread B executes withdraw(6), balance = 4
- Thread A e ... ance = 8
- Thread A e ... balance = 8
- Thread B executes depos ... ce = 11
- Thread B executes withdraw(7), balance = 4

Synchronization

Thread Synchronization

How many potential deadlock situations are present?


```
sem_t add_sem;
sem_t rem_sem;
```

Thread Synchronization (Contd.)

Situation 1:

tid1 executes V(&add_sem) and V(&rem_sem). Then, tid2 executes P(& m). In this

situation, tid1 can m) since the value of add_sem = 0. As a resu deadlock,

since after the execution of thread 2, thread 1 can’t resume. Thus, there’s a deadlock.

Thread Synchronization (Contd.)

Situation 2:

tid1 executes V(&add_sem) and V(&rem_sem). Then, tid2 executes P(&

P(&add_sem). Th P(&add_sem) but it can't since add_sem has v ead 1 wants to execute P(&rem_sem) but it can't since rem_sem has value 0. Thus, there's a deadlock.

Thread Synchronization (Contd.)

For lengths 0-6, indicate the number of outcomes of that length that can be produced.

```
sem_t add_sem;
sem_t rem_sem;

void add() {
    printf("A");
}

void remove() {
    printf("R");
}
```



```
void *thread2(void *vargp) {
    P(&rem_sem);
    P(&add_sem);

    add();
    remove();
}
```

Thread Synchronization (Contd.)

Response length 0: None

This is because at least 'R' must get printed due to the

call to remove() in

at least that state

sort of deadlock from the above

a deadlock,

before any

Thread Synchronization (Contd.)

Response length 1: 1 (R)

In the deadlock scenario 2, where thread 1 executes P(&add_sem) an

neither of the thre at. Thus, no print statements are executed in ad after that point. The only print statement that gets executed is due to the call to remove() before the calls to P() in thread 1.

Thread Synchronization (Contd.)

Response length 2: None

We noticed that 'R' due to the call to remove() in
ne code, we thread1() gets pri

notice that it's not her print
statement to get executed.

Thread Synchronization (Contd.)

Response length 3: 2 (RAR, ARR)

This happens due to deadlock scenario 1 above, where

) can't thread2() execute

after that. execute P(&add_

- RAR: Thread 1 executes remove(), followed by thread 2 executing add() and remove().
- ARR: Thread 2 executes add(), followed by any ordering of the 2 calls to remove() by threads 1 and 2.

Thread Synchronization (Contd.)

Response length 4: None

For any length greater than 3, it means that there was no deadlock, since th

etion and

ich thread 1

means it could get

would run to completion as well.

sponses of length greater than 3 and less than 6 are possible.

Thread Synchronization (Contd.)

Response length 5: None

For any length greater than 3, it means that there was no

etion and deadlock, since th

ich means it could get thread 1

would run to completion as well. sponses of

length greater than 3 and less than 6 are possible.

Thread Synchronization (Contd.)

Response length 6: 4
(RARAAR, RARARA, RAARRA, RAARAR)

ne inst Since there are n calls to V() and P(d 1. Thus, ‘R’ and ‘A’ definitely get printed. Aft calls to V() get executed by thread 1 and then, thread 2 can execute its calls to P(). After this, based on the interleavings between the threads, there are 4 possible outputs.

Thread Synchronization (Contd.)

- RARAAR: Thread 1 executes remove(), threads 1 and 2 execute the add() statements in any order, and then thread 2 executes remove().

- RARARA: Thr (), thread 2 executes add() and remove() d 1 executes add().

Thread Synchronization (Contd.)

- RAARRA: Thread 2 executes add(), threads 1 and 2 execute the remove() statements in any order, and then thread 1 executes add().

- RAARAR: Thre d 1 executes remove() and add(), then threa remove().

Good luck!

Processes

1. Logical control flow
2. Private address spa

Important system calls

1. Fork
2. Execve
3. Wait
4. Waitpid

Processes

Draw a Process Graph!!!

(it does not have to be like mine)

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
         count += 2;
    }
    printf(“%d”, count);
}
```

What is printed?

Assume printf is atomic,

ystem calls

.

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
         count++;
    else{
         if(pid2 == 0)
              count--;
         else
           count += 2;
    }
    printf("%d", count);
}
```

How many processes?

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```

How many processes?

Parent: forks child

nd child: each fork
child

cesses

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
         count++;
    else{
         if(pid2 == 0)
               count--;
         else
           count += 2;
    }
    printf("%d", count);
}
```

What does the process diagram look like?

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
         count++;
    else{
         if(pid2 == 0)
              count--;
         else
           count += 2;
    }
    printf("%d", count);
}
```

What does the process diagram look like?

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();
    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
          count += 2;
    }
    printf("%d", count);
}
```

d2 != 0

id2 != 0

d2 == 0

What does count look like?

Grandchild: pid2 == 0

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();
    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
            count += 2;
    }
    printf("%d", count);
}
```

d2 != 0

id2 != 0

What does count look like?

- co

Child2: pid == 0

- count = 0

Grandchild: pid1 == 0 and pid2 == 0

- count = 2

Processes

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
          count += 2;
    }
    printf("%d", count);
}
```

Given the process diagram, what are the different permutations that can be printed out?

Processes

Given the process diagram, what are the different permutations that can be printed out?

```
int main() {
    int count = 1;
    int pid1 = fork();
    int pid2 = fork();

    if(pid1 == 0)
        count++;
    else{
        if(pid2 == 0)
            count--;
        else
          count += 2;
    }
    printf("%d", count);
}
```


Math! 4! / 2 = 12 different possible outcomes

Processes

Remember:

- Processes can occur in any order
  - atch out for a `wait` or `waitpid`!
  - What if I included a `wait(NULL)` before I printed out count?
- Good luck!

File IO

How the Unix Kernel Represents Open Files

File IO

```
foo.txt: abcdefgh...xyz
int main() {
     int fd1, fd2, fd3;
     char c;
     pid_t pid;
     fd1 = open("foo.txt", O_RDONLY);
     fd2 = open("foo.txt",
     fd3 = open("foo.txt",
     read(fd1, &c, sizeof(
     read(fd2, &c, sizeof(c));          // c = ?
     dup2(fd2, fd3);
     read(fd3, &c, sizeof(c));          // c = ?
     read(fd2, &c, sizeof(c));          // c = ?
```

Main ideas:

- How does read offset?
- How does dup2 work?
  - What is the order of arguments?
  - d3 share offset with fd2?

File IO

```
foo.txt: abcdefgh...xyz
int main() {
      int fd1, fd2, fd3;
      char c;
      pid_t pid;
      fd1 = open("foo.txt", O_RDONLY);
      fd2 = open("foo.txt",
      fd3 = open("foo.txt",
      read(fd1, &c, sizeof(
      read(fd2, &c, sizeof(c));         // c = a
      dup2(fd2, fd3);
      read(fd3, &c, sizeof(c));         // c = b
      read(fd2, &c, sizeof(c));         // c = c
```


- How does read offset?
  - Incremented by number of bytes read
- How does dup2 work?

ad/write from fd3 now

n from fd2

sets are shared

File IO

```
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c
```

```
pid = fork();
if (pid==0) {
     read(fd1, &c,
     printf("c = %c\n", c);
     dup2(fd1, fd2);
     read(fd3, &c, sizeof(c));
     printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
```

Main ideas:

- How are fd shared between processes?
- How does dup2 work from parent

le offsets shared
esses?

File IO

```
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
     read(fd1, &c,
     printf("c = %c\n", c);
     dup2(fd1, fd2);
     read(fd3, &c, sizeof(c));
     printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
```


What would this program print?

possible outcomes due

to i try two simple cases :

1. First child executes to the end
2. First parent executes to the end.

File IO

```
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
     read(fd1, &c,
     printf("c = %c\n", c);
     dup2(fd1, fd2);
     read(fd3, &c, sizeof(c));
     printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
```

Possible output 1:

c = b // in child

c = d // in child

hild

hild

ent

ent

File IO

```
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
     read(fd1, &c,
     printf("c = %c\n", c);
     dup2(fd1, fd2);
     read(fd3, &c, sizeof(c));
     printf("c = %c\n", c);
}
read(fd2, &c, sizeof(c));
printf("c = %c\n", c);
read(fd1, &c, sizeof(c));
printf("c = %c\n", c);
```

Possible output 2:

c = d // in parent

c = b // in parent

hild from fd1

hild from fd3

d

d

File IO

```
    .
    .
    pid = fork();
    if (pid==0) {
        read(fd1, &c, sizeof(c));
        printf("c = %c
        dup2(fd1, fd2)
        read(fd3, &c,
        printf("c = %c\n", c);
    }
    if (pid!=0) waitpid(-1, NULL, 0);
    read(fd2, &c, sizeof(c));
    printf("c = %c\n", c);
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    return 0;
}
```

he possible outputs now?

File IO

```
    .
    .
    pid = fork();
    if (pid==0) {
        read(fd1, &c, sizeof(c));
        printf("c = %c
        dup2(fd1, fd2)
        read(fd3, &c,
        printf("c = %c\n", c);
    }
    if (pid!=0) waitpid(-1, NULL, 0);
    read(fd1, &c, sizeof(c));
    printf("c = %c\n", c);
    read(fd2, &c, sizeof(c));
    printf("c = %c\n", c);
    return 0;
}
```

Possible output:

c = b // in child

c = d // in child

hild

hild

ent

ent

File IO

```
read(fd1, &c, sizeof(c)); // a
read(fd2, &c, sizeof(c)); // a
dup2(fd2, fd3);
read(fd3, &c, sizeof(c)); // b
read(fd2, &c, sizeof(c)); // c

pid = fork();
if (pid==0) {
     read(fd1, &c,
     dup2(fd1, fd2);
     read(fd3, &c, sizeof(c));
}
if (pid!=0) waitpid(-1, NULL, 0);
read(fd2, &c, sizeof(c));
read(fd1, &c, sizeof(c));
```

- Child creates a copy of the parent fd table
  - dup2/open/close in parent affect the child
  - open/close in child do affect the parent
- rs across process me file offset.

Malloc

Malloc

- Fit algorithms - first/next/best/good
- Fragmentation
  - Internal - inside blocks
  - External -
- Organization
  - Implicit
  - Explicit
  - Segregated

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)						
b = malloc(16)						
c = malloc(16)						
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

final-f01.pdf: Problem 10 Part A

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)						
c = malloc(16)						
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
d = malloc(16)						
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

final-f01.pdf: Problem 10 Part A

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

final-f01.pdf: Problem 10 Part A

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)						
f = malloc(48)						
free(b)						

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)	48a	32a	80f [0]			
f = malloc(48)						
free(b)						

final-f01.pdf: Problem 10 Part A

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)	48a	32a	80f [0]			
f = malloc(48)	48a	32a	80a			
free(b)						

final-f01.pdf: Problem 10 Part A

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)	48a	32a	80f [0]			
f = malloc(48)	48a	32a	80a			
free(b)	48a	32f [0]	80a			

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal?

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)	48a	32a	80f [0]			
f = malloc(48)	48a	32a	80a			
free(b)	48a	32f [0]	80a			

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - (48-16) + (80-48) = 64
  - external?

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)	48a	32a	80f [0]			
f = malloc(48)	48a	32a	80a			
free(b)	48a	32f [0]	80a			

Malloc - First fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - (48-16) + (80-48) = 64
  - external
  - 32

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48a	32a	32f [0]	48a		
free(d)	48a	32a	80f [0]			
f = malloc(48)	48a	32a	80a			
free(b)	48a	32f [0]	80a			

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)						
b = malloc(16)						
c = malloc(16)						
free(a)						
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)						
free(d)						
f = malloc(48)						
free(b)						

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)						
f = malloc(48)						
free(b)						

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)	48f [1]	32a	32a	48f [0]		
f = malloc(48)						
free(b)						

final-f01.pdf: Problem 10 Part A

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
[illegible]	[illegible]	[illegible]	32a	48a		
[illegible]	[illegible]	[illegible]	32f [0]	48a		
free(a)	4[illegible]	[illegible]	[illegible]2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)	48f [1]	32a	32a	48f [0]		
f = malloc(48)	48f [0]	32a	32a	64a		
free(b)						

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4[illegible]		[illegible]2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)	48f [1]	32a	32a	48f [0]		
f = malloc(48)	48f [0]	32a	32a	64a		
free(b)	80f [0]		32a	64a		

final-f01.pdf: Problem 10 Part A

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal?

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)	48f [1]	32a	32a	48f [0]		
f = malloc(48)	48f [0]	32a	32a	64a		
free(b)	80f [0]		32a	64a		

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - (32-16) + (64-48) = 32
  - external?

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)	48f [1]	32a	32a	48f [0]		
f = malloc(48)	48f [0]	32a	32a	64a		
free(b)	80f [0]		32a	64a		

Malloc - Best fit

- 16 byte align
- coalesced
- footerless
- 32 min size
- fragmentation?
  - internal
  - (32-16) + (64-48) = 32
  - external
  - 80

	#1	#2	#3	#4	#5	#6
a = malloc(32)	48a					
b = malloc(16)	48a	32a				
c = malloc(16)	48a	32a	32a			
			32a	48a		
			32f [0]	48a		
free(a)	4		2f [1]	48a		
e = malloc(16)	48f [0]	32a	32a	48a		
free(d)	48f [1]	32a	32a	48f [0]		
f = malloc(48)	48f [0]	32a	32a	64a		
free(b)	80f [0]		32a	64a		

Signals

who would win?

several hundre
tshlab c

```
void
exec_cmdline(char *cmdline, char **argv, sigset_t *set,
        int bg, int fd_in, int fd_out)
{
    pid_t pid = 0;
    if ((pid = Fork()) == 0) {
        // Child process; restore mask and execute job.
        Sigprocmask(SIG_SETMASK, set, NULL);
```

ynchronous boi

Signals

Signals (Contd.)

- Sending the same signal to the parent in all the calls to kill() may print 1 since there would be no queuing of signals.

- We can guara[illegible]d precisely one SIGUSR1 and one SIGUSR2
- We can print 1-4 depending on the manner in which signals are sent and received.