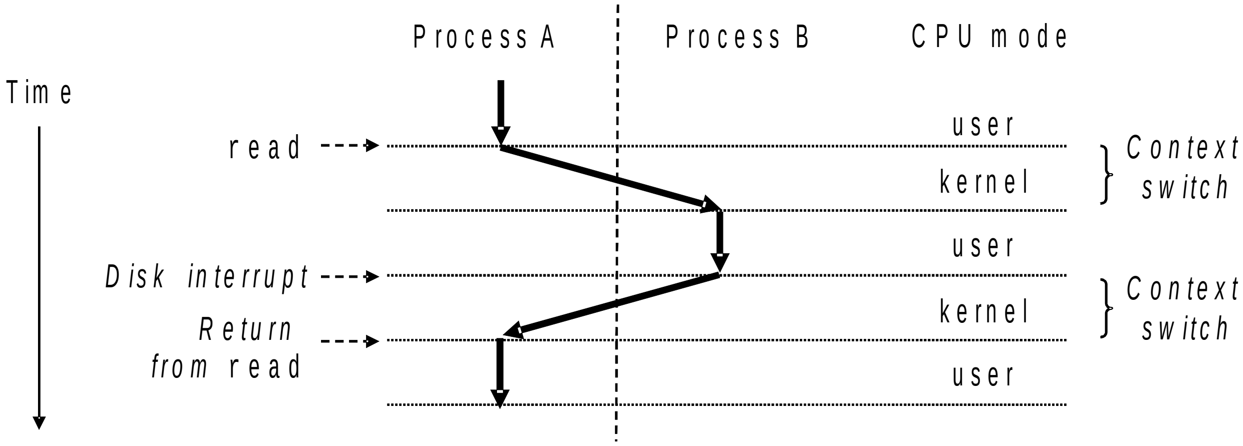
Sample Solution

Question 1
Concurrently Running Processes

Modern operating systems provide *time-shared* multitasking. In time-shared multitasking, one CPU is shared among several processes. In the following diagram, draw the control flow and write down in which mode the CPU is running.



Question 2
Implementation of Linux/IA32 System Calls

The following table shows some of the system calls used in Linux/IA32.

%eax	Name	%ebx	%ecx	%edx
1	sys_exit	int		
2	sys_fork	struct pt_regs		
3	sys_read	unsigned int	char*	size_t
4	sys_write	unsigned int	const char*	size_t
5	sys_open	const char*	int	int
6	sys_close	unsigned int		

Consider the following program, which consists of system calls without wrappers in Linux.

```
/* myhello.c */
int main()
{
    int len;
    char buf[10];
    my_write(1, “What is your student id? :\n”, 27);
    len = my_read(2, buf, 10);
    my_write(1, “SID : “, 6);
    my_write(1, buf, len);
    my_exit(0);
}
```

When you compile and execute the above code, it will produce the following result.

```
$ gcc -o my_hello -m32 myhello.c mylib.s
$ ./my_hello
What is your student id? :
2015-12345
SID : 2015-12345
$
```

Write the assembly program that supports the above result. (follow calling conventions)

```
# mylib.s
.section .text
.globl my_write
.globl my_read
.globl my_exit


# void my_write(unsigned int fd,
                const char *string,
                size_t length);
my_write:
  push %ebp
  mov  %esp, %ebp
  push %ebx

  mov  $4, %eax
  mov  8(%ebp), %ebx
  mov  12(%ebp), %ecx
  mov  16(%ebp), %edx

  int  $0x80

  pop  %ebx
  pop  %ebp

  ret
```

```
# void my_read(unsigned int fd,
               char *buffer,
               size_t length);
my_read:
  push %ebp
  mov  %esp, %ebp
  push %ebx

  mov  $3, %eax
  mov  8(%ebp), %ebx
  mov  12(%ebp), %ecx
  mov  16(%ebp), %edx

  int  $0x80

  pop  %ebx
  pop  %ebp

  ret


# void my_exit(int nr);
my_exit:
  push %ebp
  mov  %esp, %ebp
  push %ebx

  mov  $1, %eax
  mov  8(%ebp), %ebx

  int  $0x80

  pop  %ebx
  pop  %ebp

  ret
```

Question 3

Virtual Memory Address Spaces

Complete the following table, filling in the missing entries and replacing each question mark with the appropriate integer. Use the following units: *$K = 2^{10}$ (Kilo), $M = 2^{20}$ (Mega), $G = 2^{30}$ (Giga), $T = 2^{40}$ (Tera), $P = 2^{50}$ (Peta), or $E = 2^{60}$ (Exa).*

# virtual address bits (n)	# virtual address (N)	Largest possible virtual address
8	256	255
16	$2^?$ = 64K (? = 16)	2^{16} - 1
32	4G	2^{32} – 1 = ?G – 1 (? = 4)
48	$2^?$ = 256T (? = 48)	2^{48} - 1
64	16E	2^{64} - 1

Question 4

Address Translation

Given a 32-bit virtual address space and a 24-bit physical address, determine the number of bits in the *virtual page number (VPN), virtual page offset (VPO), physical page number (PPN), physical page offset (PPO)* for the following page sizes *P*:

P	# VPN bits	# VPO bits	# PPN bits	# PPO bits
1 KB	22	10	14	10
2 KB	21	11	13	11
4 KB	20	12	12	12
8 KB	19	13	11	13