**Pınar Yücel**
**21802188**

# CS342 Operating Systems - Spring 2021
# Homework #2

---

1. C program that creates $2^k$-1 processes:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>

void createTree(int k)
{
        if(k == 1)
        {
                return;
        }

        pid_t n1 = fork();

        if(n1 > 0)
        {
                pid_t n2 = fork();

                if(n2 > 0) // parent
                {
                        wait(NULL);
                        wait(NULL);
                }
                else if(n2 == 0) // left child
                {
                        createTree(k - 1);
                }
        }
        else if(n1 == 0) // right child
        {
                createTree(k - 1);
        }
}

int main()
{
        int k = -1;

        while(k > 5 || k < 1)
        {
                printf("Please enter a value between 1 and 5\n");
                printf("k: ");
                scanf("%d", &k);
                printf("\n");
        }

        createTree(k);

        printf("Process id: %d\n", getpid());

        return 0;
}
```

2. Names of 10 fields:
    - volatile long state
    - unsigned int flags
    - unsigned int ptrace
    - unsigned int cpu
    - unsigned int wakee_flips
    - unsigned long wakee_flip_decay_ts
    - int on_rq
    - int prio
    - int static_prio
    - int normal_prio

3. Assuming that initially the only process was the initial main program, **4** processes will remain after the execution of the given pseudo-code. However, in general, if n processes execute the given pseudo-code, **4n** processes will remain.

4. The following integers will be printed:
    - 100
    - 200
    - 200
    - 200
    - 250
    - 250
    - 250

5. C program that runs commands:

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/types.h>
#include <unistd.h>
```

```
#include <sys/wait.h>

int main()
{
	pid_t n1 = fork();

	if(n1 > 0)
	{
		pid_t n2 = fork();

		if(n2 > 0) // parent
		{
			wait(NULL);
			wait(NULL);
		}
		else if(n2 == 0) // left child
		{
			execlp("/bin/ps", "ps", "aux",  NULL);
		}
	}
	else if(n1 == 0) // right child
	{
		execlp("/bin/ls", "ls", "-al",  NULL);
	}

	return 0;
}
```

6. Run the producer first.

   producer.c

```
#include <stdlib.h>
#include <mqueue.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include "shareddefs.h"

int main()
{
	mqd_t mq;
	struct item item;
```

```
        int n;

        mq = mq_open(MQNAME, O_RDWR | O_CREAT, 0666, NULL);

        if (mq == -1)
        {
                perror("mq_open failed\n");
                exit(1);
        }

        strcpy(item.astr, "I hear and I forget. I see and I remember. I do and I
understand.");
        n = mq_send(mq, (char *) &item, sizeof(struct item), 0);

        if (n == -1)
        {
                perror("mq_send failed\n");
        exit(1);
        }

        mq_close(mq);
        return 0;
}
```

consumer.c

```
#include <stdlib.h>
#include <mqueue.h>
#include <stdio.h>
#include <unistd.h>
#include <errno.h>
#include <string.h>
#include "shareddefs.h"

int main()
{       mqd_t mq;
        struct mq_attr mq_attr;
        struct item *itemptr;
        int n;
        int buflen;
        char *bufptr;

        mq = mq_open(MQNAME, O_RDWR | O_CREAT, 0666, NULL);
```

```
	if (mq == -1)
	{
		perror("can not create msg queue\n");
		exit(1);
	}

	mq_getattr(mq, &mq_attr);

	/* allocate large enough space for the buffer */
	buflen = mq_attr.mq_msgsize;
	bufptr = (char *) malloc(buflen);

	n = mq_receive(mq, (char *) bufptr, buflen, NULL);

	if (n == -1)
	{
		perror("mq_receive failed\n");
		exit(1);
	}

	itemptr = (struct item *) bufptr;
	printf("%s\n", itemptr->astr);

	free(bufptr);
	mq_close(mq);
	return 0;
}
```

shareddefs.h

```
struct item {
	char astr[100];
};


#define MQNAME "/justaname"
```

Makefile

```
all: producer consumer
```

```
consumer: consumer.c
	gcc -Wall -o consumer consumer.c -lrt

producer: producer.c
	gcc -Wall -o producer producer.c -lrt

clean:
	rm -fr *~ producer consumer
```

7. C program that copies a file:

```
#include <stdio.h>
#include <sys/types.h>
#include <unistd.h>
#include <sys/wait.h>
#include <fcntl.h>

int main()
{
	int fd1 = open("file", O_RDONLY);
	int fd2 = open("anotherFile", O_WRONLY| O_CREAT, 00700);

	char c;

	while (read(fd1, &c, 1) == 1)
	{
		write(fd2, &c, 1);
		write(fd2, &c, 1);
	}

	close(fd1);
	close(fd2);

	return 0;
}
```