210011 Dodajon Xusnitdinov

task 1

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

int main()
{
	pid_t   pid;
	printf("fork program starting \n");
	pid = fork();
	switch(pid){
	case -1: perror("fork failed"); break;
	case 0: for(int i = 0; i<=9; ++i){
	printf("%d",i);
	} break;
	default: for(char c='A'; c<='Z'; ++c){
	printf("%c",c);
	}; break;
	};

	return 0;
}
```

task 2

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int main() {
	printf("Executing ls command...\n");

	// Constructing argument vector
	char *args[] = {"/bin/ls", "/bin", "/home" , NULL}; // Command to execute

	execv(args[0], args); // Execute the command

	// If execv returns, it means there was an error
	perror("execv");
	return 1;
}
```

task 3

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

int call_exec(){
  char *args[] = {"./child", NULL};
  execv(args[0],args);
};

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

  switch (pid){
  case -1:  perror("fork failed"); break;
  case 0: {
   printf("Child PID: %d\n", getpid());
   call_exec();
   break;
  }
  default: printf("Parent PID: %d\n", getpid()); break;
}



  exit(0);
}
```

task 4

```
#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>

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

  switch(pid){
   case -1: perror("fork failed");
   case 0: {
           printf("This is child before execl %d\n", getpid());
           execl("/usr/bin/ps", "ps", "-l", NULL);
           break;
   };
   case 1: {
           printf("This is parent %d\n", getpid());
           printf("This is child from parent process %d\n", pid);
   };
 };

  exit(0);
}
```

task 5

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

int main() {
        pid_t pid;

        pid = fork();

        switch (pid) {
        case -1: // fork failed
        perror("fork failed");
        exit(1);
        break;
        case 0: // Child process
        sleep(2); // Sleep for 2 seconds to ensure the parent process starts first
        printf("Child process killing parent process %d\n", getpid(), getppid());
        kill(getppid(), SIGKILL); // Sending SIGKILL signal to parent
        break;
        default: // Parent process
        printf("Parent process is running.%d\n", getpid());
        printf("Parent process waiting for child process to terminate… %d\n", getpid());
        wait(NULL); // Wait for child process to terminate
        printf("Parent process terminated by child. %d\n", getpid());
        }

        return 0;
}
```

task 6

```
#include <stdio.h>
#include <signal.h>
#include <unistd.h>
```

```
void signal_handler(int sig){
        printf("Signal received: %d\n", sig);
}       //handling ctrl + c and ctrl + v

int main(){
        signal(SIGINT, signal_handler);
        signal(SIGTSTP, signal_handler);

        while(1){
        printf("Hello World!\n");
        sleep(1);
        }
}
```

This is all I could do by myself, but ran out of time for the rest of the problems