Problem statement:

Implement a parking management application. This should have functions which will simulate the arrival of vehicles to the parking lot (arrival()) and departure of vehicles from the parking lot (departure()). These functions should run on separate processes. The application should define the total parking lots in the parking place and make sure that vehicles are allowed only when there is free parking lot is available. Similarly, the application should not allow the departure function to if no vehicles are there in the parking place.

Child and Parent process

- Child process: A child process is a process created by a parent process in operating system using a fork() system call. A child process is created as its parent process's copy and inherits most of its attributes. If a child process has no parent process, it was created directly by the kernel. The Process ID (PID) of the child process is returned to the parent process

- Parent process: All the processes in operating system are created when a process executes the fork() system call except the startup process. The process that used the fork() system call is the parent process. In other words, a parent process is one that creates a child process. A parent process may have multiple child processes but a child process only one parent process.

Pipe

- A pipe is a connection between two processes, such that the standard output from one process becomes the standard input of the other process. In UNIX Operating System, Pipes are useful for communication between related processes(inter-process communication).

Implementation

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

int n;
int arrival(int n)
{
   if (n != 0)
   {
      printf("Vehicle parked!");
      printf("\n");
      n = n - 1;
      printf("Number of parking lots available: %d", n);
      printf("\n");
      return n;
   }
   else
   {
      printf("No parking lots available!");
```

```
        printf("\n");
        return n;
    }
}

int departure(int n)
{
    if (n == 10)
    {
        printf("No vehicle for departure to happen!");
        printf("\n");
        return n;
    }
    else
    {
        printf("Vehicle departed!");
        printf("\n");
        n = n + 1;
        printf("Number of parking lots available: %d", n);
        printf("\n");
        return n;
```

```
    }
}
int main()
{
    n = 10;
    int x;

    int p[2];
    int returnstatus;
    int pid, pid1;
    int readmessage[1];

    returnstatus = pipe(p);

    if (returnstatus == -1)
    {
        printf("Unable to create pipe\n");
        return 1;
    }
    pid = fork();
```

```
if (pid == -1)
{
    return 1;
}

// child process 1
if (pid == 0)
{
    int c;
    if (read(p[0], & c, sizeof(c)) == -1)
    {
        return 3;
    }
    n = arrival(n);

    if (write(p[1], & c, sizeof(c)) == -1)
    {
        return 4;
    }
}
```

```
// child process 2
if (pid1 == 0)
{
    int c;
    if (read(p[0], & c, sizeof(c)) == -1)
    {
        return 5;
    }
    n = departure(n);
    if (write(p[1], & c, sizeof(c)) == -1)
    {
        return 6;
    }
}

// parent process
else
{
    do
    {
        printf("Enter your choice: ");
```

```
printf("\n");
printf("1. Arrival");
printf("\n");
printf("2. Departure");
printf("\n");
printf("3. Exit");
printf("\n");
scanf("%d", & x);
printf("\n");
switch (x)
{
case 1:
    if (write(p[1], & x, sizeof(x)) == -1)
    {
        return 7;
    }
    n = arrival(n);
    if (read(p[0], & x, sizeof(x)) == -1)
    {
        return 8;
    }
```

```
            break;
        case 2:
            if (write(p[1], & x, sizeof(x)) == -1)
            {
                return 9;
            }
            n = departure(n);
            if (read(p[0], & x, sizeof(x)) == -1)
            {
                return 10;
            }
            break;
        case 3:
            exit(0);
        default:
            printf("Invalid choice!");
            printf("\n");
        }
    }
    while (x != 1 || x != 2 || x != 3);
}
```

```
    close(p[0]);
    close(p[1]);

    return 0;
}
```

- **Code Screenshots:**

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int n;
int arrival(int n)
{
     if(n != 0)
     {
          printf("Vehicle parked!");
          printf("\n");
          n = n - 1;
          printf("Number of parking lots available: %d", n);
          printf("\n");
          return n;
     }
     else
     {
          printf("No parking lots available!");
          printf("\n");
          return n;
     }
}

int departure(int n)
{
     if(n == 10)
     {
          printf("No vehicle for departure to happen!");
          printf("\n");
          return n;
     }
     else
     {
          printf("Vehicle departed!");
          printf("\n");
          n = n + 1;
          printf("Number of parking lots available: %d", n);
          printf("\n");
          return n;
     }
}

```

```c
int main()
{
    n = 10;
    int x;

    int p[2];
    int returnstatus;
    int pid, pid1;
    int readmessage[1];

    returnstatus = pipe(p);

    if (returnstatus == -1)
    {
        printf("Unable to create pipe\n");
        return 1;
    }
    pid = fork();

    if(pid == -1)
    {
        return 1;
    }

    // child process 1
    if (pid==0)
    {
        int c;
        if(read(p[0], &c, sizeof(c)) == -1)
        {
            return 3;
        }
        n = arrival(n);

        if(write(p[1], &c, sizeof(c)) == -1)
        {
            return 4;
        }
    }

```

```
    // child process 2
    if (pid1==0)
    {
        int c;
        if(read(p[0], &c, sizeof(c)) == -1)
        {
            return 5;
        }
        n = departure(n);
        if(write(p[1], &c, sizeof(c)) == -1)
        {
            return 6;
        }
    }

    // parent process
    else
    {
        do
        {
            printf("Enter your choice: ");
            printf("\n");
            printf("1. Arrival");
            printf("\n");
            printf("2. Departure");
            printf("\n");
            printf("3. Exit");
            printf("\n");
            scanf("%d", &x);
            printf("\n");
            switch(x)
            {
                case 1:
                        if(write(p[1], &x, sizeof(x)) == -1)
                        {
                            return 7;
                        }
                        n = arrival(n);
                        if(read(p[0], &x, sizeof(x)) == -1)
                        {
                            return 8;
                        }
                        break;
                case 2:
                        if(write(p[1], &x, sizeof(x)) == -1)
                        {
                            return 9;
                        }
                        n = departure(n);
                        if(read(p[0], &x, sizeof(x)) == -1)
                        {
                            return 10;
                        }
                        break;
                case 3:
                        exit(0);
                default:
                        printf("Invalid choice!");
                        printf("\n");
            }
        }
        while(x!=1 || x!=2 || x!=3);
    }

    close(p[0]);
    close(p[1]);

    return 0;
}
```

- **Output Screenshots:**

```
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 9
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 8
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 7
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 6
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 5
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 4
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 3
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 2
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

Vehicle parked!
Number of parking lots available: 1
Enter your choice:
1. Arrival
2. Departure
3. Exit
1
```

```
Vehicle parked!
Number of parking lots available: 0
Enter your choice:
1. Arrival
2. Departure
3. Exit
1

No parking lots available!
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 1
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 2
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 3
Enter your choice:
1. Arrival
2. Departure
3. Exit
2
```

```
Vehicle departed!
Number of parking lots available: 4
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 5
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 6
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 7
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 8
Enter your choice:
1. Arrival
2. Departure
3. Exit
2
```

```
Vehicle departed!
Number of parking lots available: 9
Enter your choice:
1. Arrival
2. Departure
3. Exit
2

Vehicle departed!
Number of parking lots available: 10
Enter your choice:
1. Arrival
2. Departure
3. Exit
2
```

```
No vehicle for departure to happen!
Enter your choice:
1. Arrival
2. Departure
3. Exit
3

sadhana@sadhana-VirtualBox:~/Documents/Operating systems/Project$ ./Project
Enter your choice:
1. Arrival
2. Departure
3. Exit
5

Invalid choice!
Enter your choice:
1. Arrival
2. Departure
3. Exit
3

sadhana@sadhana-VirtualBox:~/Documents/Operating systems/Project$
```

Conclusion

In our code, we implemented a parking management application by making an arrival() in the child process and the departure() in the parent process. We also made sure that no vehicle will be allowed to enter if the parking lot is full, and no vehicle will be allowed to depart if the parking lot is empty. We made use of pipe to communicate between the two processes.