컴퓨터학과
20162449
김상현

1. 구현 개요

세포 매트릭스 게임 구현.

- 각 세대 별로 생기는 세포의 결과를 파일에 출력 한다.
- 세대 규칙은 다음과 같다

1. 해당 세포가 살아있을 때, 살아있는 이웃 세포의 수가 2개 이하 혹은 7개 이상인 경우 해당 세포는 다음 세대에 죽는다.
2. 해당 세포가 살아있을 때, 살아있는 이웃 세포의 수가 3개 ~ 6개인 경우 해당 세포는 다음 세대에 살아서 유지된다.
3. 해당 세포가 죽어있을 때, 살아있는 이웃 세포의 수가 4개인 경우 해당 세포는 다음 세대에 살아서 유지된다.
4. 해당 세포가 죽어있을 때, 살아있는 이웃 세포의 수가 4개를 제외한 나머지 경우들에는 해당 세포는 다음 세대에 여전히 죽어있는 상태가 유지된다.

세포 매트릭스 게임 함수 : void game(int m, int n, char **, int start, int end, char *)

- 인자로 받은 matrix 2차원 배열을 가지고 이웃의 살아있는 세포의 개수를 저장하고 위의 규칙에 따라 **charbuffer를 만든다. 다음 세대의 세포 matrix를 저장한 charbuffer를 인자로 받은 filename 파일에 출력한다.
- 인자로 받은 start 와 end 는 charbuffer를 정할 때 사용한다. 즉 start 행이 0 , end 행이 4행이라면 0~4행까지의 다음 세대의 세포 매트릭스를 charbuffer에 저장한다. (charbuffer은 (end - start) 의 행과 n개의 열로 구성 되어진 2차원배열이다.

3가지 방식으로 구현

(순차처리 , 멀티 프로세스를 이용한 병렬처리 , 멀티 스레드를 이용한 병렬처리)

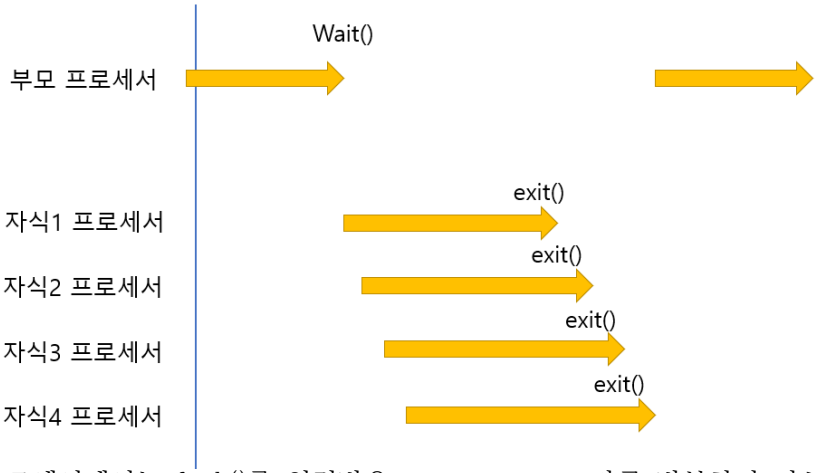
1. 순차 처리 :

game() 함수를 호출한다. 이때 start는 0 ~ end는 끝 행 으로 설정하여 한세대에 한번 game 함수를 호출하게 된다.

자세한 설명은 생략하겠다.

2. 멀티 프로세서를 이용한 병렬처리

기본 개념을 그림으로 그려보면 다음과 같다.



부모 프로세서에서는 fork()를 입력받은 process_num 만큼 반복하여 자식 프로세서를 생성한다. (자식 프로세서는 부모프로세서의 변수와 함수들을 복사하기는 하지만 다른 통신 기법이 없다면 자식 프로세서에서 부모프로세서의 변수 등을 수정할 수 없다. 즉 메모리 공유가 되지 않는다.)

부모 프로세서는 모든 자식 프로세서가 완료 (exit()) 가 될 때까지 wait() 함수를 통해 블록처리한다. 모든 자식 프로세스가 끝나게 되면 부모 프로세서는 종료된다.

```
334             for (int id = 0;id<process_num;id++){
335                 if((child_pid[i][id] = fork())==0){
336                     jump = count_m / count_process;
337                     count_m -= jump;
338                     count_process -=1;
339                     game(m,n,temp_matrix,start,start+jump,filename);
340                     start+=jump;
341                     exit(0);
342                 }
343                 else if(child_pid[i][id] >0){
344                     jump = count_m /count_process;
345                     count_m -=jump;
346                     count_process-=1;
347                     start+=jump;
348                 }
349             }
```

- fork()를 통한 자식 프로세서 , fork() 의 리턴 값이 0 인경우는 자식 프로세서에서 실행된다는 뜻이기 때문에 for 문안에 if 문은 자식프로세서에서 , else if 문은 부모 프로세서에서 실행된다. 부모프로세서에서는 자식 프로세서에서 진행된 증감 연산을 똑같이 수행함으로써 메모리가 공유되지않아 생기는 문제를 해결 했다.

- 한 세대에 한번의 game 호출이 일어나지 않고 각 자식프로세서들마다 game 함수를 호출하게 되는 이 때에 game 함수의 인자가될 행의 시작과 끝을 어떻게 균등히 배분 하였는지에 대해 알아보자.

● 각 자식 프로세서들에게 균등히 행을 배분 하는 문제 해결 방법

- 쉽게 예를 들어 설명해 보려고 한다. 전체 행이 50개 프로세서 가 5개라고 한다면 .

1) 전체 행을 프로세서의 수로 나눈 값을 start 행(처음 start는 당연히 0부터 시작이다.)과 end 행의 사이 값(이하 jump)으로 지정한다.
ex) 50 / 5 = 10 , start = 0 , end = start + 10

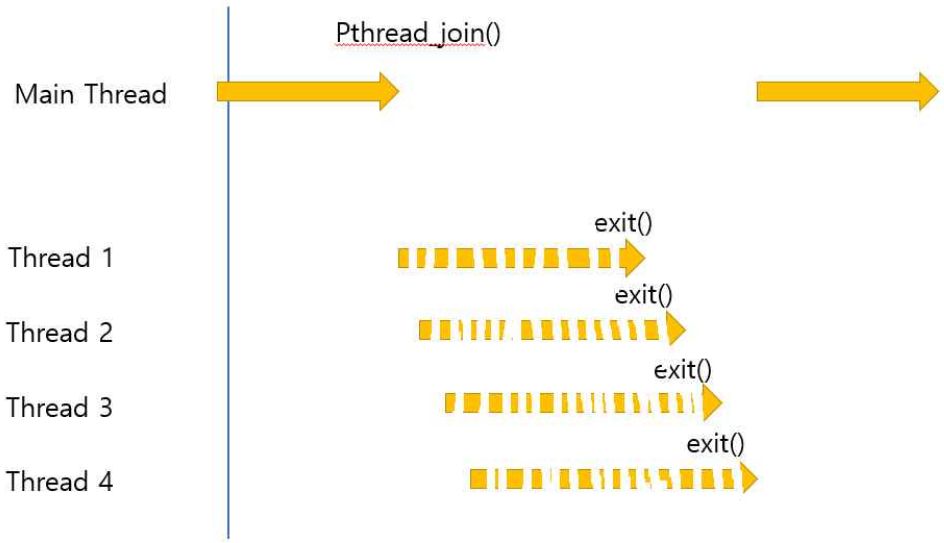
2) 전체 행의 수에서 사잇 값(jump)을 빼고, process 수도 한 개 뺀다.
ex) 50 - 10 = 40 , 5 -1 =4

3) 처음 start 값에서 사잇 값 만큼을 더하고 저장한다.
.. 이 과정을 반복하면
start :0 end :10 -> 40 / 4 = 10(jump), 40 -10 = 30 (전체 행) 4 -1 = 3(프로세서 수)
start :10 end :20 -> 30 / 3 = 10(jump), 30 -10 = 20(전체 행) 3 -1 = 2(프로세서 수)
..
start : 40 end :50 -> 10 / 1 = 10(jump), 10 -10 = 0(전체 행) 1- 1 = 0(프로세서 수)

즉 5개의 프로세서가 각각 10개의 행 만큼 담당하게 된다.
효과적으로 전체 행을 주어진 프로세스 들에게 균등히 분배하였다.

3. 멀티스레드를 이용한 병렬처리

기본개념을 그림으로 그려보면 다음과 같다.



프로세스를 이용한 병렬처리와 비슷 해보이지만 제일 큰 차이점은 스레드를 이용한 병렬처리는 한 프로세서 내에서 처리된다는 것이다. 스레드들은 각각 OS를(커널)을 통해 실행 시간을 균등히 배분받고 사용자 입장에서는 거의 동시에 실행되고있다고 느끼게 한다. 물론 실행 순서를 아주 짧은 시간 내에 바꾸어가면서 실행되고 있지만 말이다. 또 같은 프로세스 내에서 동작하기 때문에 메모리 공유가 가능하다.

실행 되고 있는 스레드를 main Thread 라고 한다면 자식 쓰레드를 입력된 Thread_num 만큼 pThread_create()를 통해 만든다. pThread_create() 함수 내에서는 쓰레드가 만들어지면 실행될 함수와 그 인자를 받게 되는데 이 때의 함수를 thread_func() 라고 정하고 필요 인자들을 구조체 배열을 통해 넘겨 주었다.

```
int err;
printf("\n%dst gen\n",i+1);
for(int j=0;j<thread_num;j++){
   st[j].m = m;
   st[j].n = n;
   //st[j].matrix = (char **)malloc(sizeof(
   st[j].matrix = temp_matrix;
   st[j].start = start;
   jump = count_m / count_thread;
   count_m -=jump;
   count_thread -= 1;
   st[j].end = start + jump;
   strcpy(st[j].filename,filename);
   start+=jump;
   //printf("* m : %d n : %d start : %d end : %d filename : %s\n",st[j].m,st[j].n,st[j].start,st[j].end,st[j].filename);
   if(err = pthread_create(&tid[j],NULL,thread_func,(void *)&st[j])){
      fprintf(stderr,"error in thread\n");
      exit(1);
   }
// pthread_detach(tid[j]);
}
start =0;
jump =0;
count_m = m;
count_thread = thread_num;
for(int j=0;j<thread_num;j++)
   while((pthread_join(tid[j],NULL))>0);
```

※ Thread_func 는 인자로 들어온 구조체의 멤버를 game() 함수의 인자로 주고 호출하기만 하는 전달 책 역할을 하는 함수이다.

```
22 void *thread_func(void *p){
23    THREAD *pp = (THREAD *)p;
24    printf("%ld\t",pthread_self());
25    game(pp->m,pp->n,pp->matrix,pp->start,pp->end,pp->filename);
26 }
```

그리고 쓰레드 생성 때 생긴 번호를 저장한 tid 배열을 비교하고 pthread_join을 이용하여 모든 thread 가 종료될 때까지 기다린다.

2. 소스 코드(다음 페이지)

```
#include <stdio.h >
#define MAXSIZE 10000
#include <stdlib.h >
#include <string.h >
#include <unistd.h >
#include <sys /wait.h >
#include <time.h >
#include <pthread.h >
void game(int , int , char ** , int ,int, char *);
void *thread_func(void *);
typedef struct THREAD{
        int m;
        int n;
        char **matrix;
        int start;
        int end;
        char *filename;
}THREAD;
void *thread_func(void *p){
        THREAD *pp = (THREAD *)p;
        printf("%ld₩t",pthread_self());
        game(pp->m,pp ->n,pp ->matrix,pp ->start,pp ->end,pp ->filename);
}
void game(int m,int n,char **matrix,int start,int end,char *filename){
        /*for(int i=0;i<m;i++){
                for(int j=0;j<n;j++){
                        printf("%c",matrix[i][j]);
                }printf("₩n");
        }
        printf("₩n");*/
        char **charbuffer;
        charbuffer = (char **)malloc(sizeof(char *) * (end -start));
        for(int i =0;i <(end -start);i ++){
                charbuffer[i] = (char *)malloc(sizeof(char ) * n);
                memset(charbuffer[i],0,sizeof(charbuffer[i]));
        }
        int count =0;
        for(int i =start,index =0;i <end;i ++,index ++){
                for(int k =1;k <n;k +=2){
                        charbuffer[index][k] =' ';
                }
                for(int j =0;j <n;j +=2){
                        if(i -1 >0 && j -2 >0  && i +1 <m && j +2 <n){
                                if(matrix[i -1][j -2] =='1'){
                                        count ++;
                                }
                                if(matrix[i -1][j] =='1'){
                                        count++;
                                }
                                if(matrix[i -1][j +2]=='1'){
                                        count++;
                                }
```

```
                if(matrix[i][j -2]=='1'){
                        count++;
                }
                if(matrix[i][j +2]=='1'){
                        count++;
                }
                if(matrix[i +1][j -2]=='1'){
                        count++;
                }
                if(matrix[i +1][j]=='1'){
                        count++;
                }
                if(matrix[i +1][j +2]=='1'){
                        count++;
                }
        }
        else {
                if(i ==0){
                        if(j ==0){
                                if(matrix[i][j +2] =='1')
                                        count++;
                                if(matrix[i +1][j +2] =='1')
                                        count++;
                                if(matrix[i +1][j] =='1')
                                        count++;
                        }
                        else if( j +2 == n){
                                if(matrix[i][j -2] =='1')
                                        count++;
                                if(matrix[i +1][j -2] =='1')
                                        count++;
                                if(matrix[i +1][j] =='1')
                                        count++;
                        }
                        else{
                                if(matrix[i][j -2] =='1')
                                        count++;
                                if(matrix[i][j +2] =='1')
                                        count++;
                                if(matrix[i +1][j -2] =='1')
                                        count++;
                                if(matrix[i +1][j] =='1')
                                        count++;
                                if(matrix[i +1][j +2]=='1')
                                        count++;
                        }
                }
                else if(i +1 == m ){
                        if(j ==0){
                                if(matrix[i][j +2] =='1')
                                        count++;
                                if(matrix[i -1][j +2] =='1')
                                        count++;
                                if(matrix[i -1][j] =='1')
```

```
                        count++;
                }
                else if( j +2 == n){
                        if(matrix[i][j -2] =='1')
                                count++;
                        if(matrix[i -1][j -2] =='1')
                                count++;
                        if(matrix[i -1][j] =='1')
                                count++;
                }
                else{
                        if(matrix[i][j -2] =='1')
                                count++;
                        if(matrix[i][j +2] =='1')
                                count++;
                        if(matrix[i -1][j -2] =='1')
                                count++;
                        if(matrix[i -1][j] =='1')
                                count++;
                        if(matrix[i -1][j +2]=='1')
                                count++;
                }
        }
        else{
                if(j ==0){
                        if(matrix[i -1][j +2] =='1')
                                count++;
                        if(matrix[i -1][j] =='1')
                                count++;
                        if(matrix[i][j +2] =='1')
                                count++;
                        if(matrix[i +1][j] =='1')
                                count++;
                        if(matrix[i +1][j +2] =='1')
                                count++;
                }
                else if( j +2 == n){
                        if(matrix[i -1][j -2] =='1')
                                count++;
                        if(matrix[i -1][j]=='1')
                                count++;
                        if(matrix[i][j -2] =='1')
                                count++;
                        if(matrix[i +1][j -2] =='1')
                                count++;
                        if(matrix[i +1][j] =='1')
                                count++;
                }
                else{
                        if(matrix[i -1][j -2] =='1')
                                count++;
                        if(matrix[i -1][j] =='1')
                                count++;
                        if(matrix[i -1][j +2] =='1')
```

```
                                        count++;
                                if(matrix[i][j -2]=='1')
                                        count++;
                                if(matrix[i][j +2] =='1')
                                        count++;
                                if(matrix[i +1][j -2] =='1')
                                        count++;
                                if(matrix[i +1][j] =='1')
                                        count++;
                                if(matrix[i +1][j +2] =='1')
                                        count++;

                                }
                        }
                }
                if(matrix[i][j] =='1'){
                        if(count >=3 && count <7){
                                charbuffer[index][j] ='1';
                        }
                        else{
                                charbuffer[index][j] ='0';
                        }
                }
                else{
                        if(count ==4 ){
                                charbuffer[index][j] ='1';
                        }
                        else{
                                charbuffer[index][j] ='0';
                        }
                }
                //printf("[%d][%d] count = %d₩n",i,j,count);
                count =0;
        }
    }
    FILE *fp;
    fp = fopen(filename,"r+");
    fseek(fp,start * (n +1),SEEK_SET);
    for(int i =0;i <(end -start);i ++){
        fwrite(charbuffer[i],n,1,fp);
        //printf("%s₩n", charbuffer[i]);
        if(end != m || (i +1)!=(end -start))
            fprintf(fp,"₩n");
    }
    //printf("₩n");
    fclose(fp);
}
int main(int argc,char *argv[]){
    FILE *fp;
    int m,n;
    int count =0;
    char buffer[MAXSIZE]={0,};
    if((fp =fopen(argv[1],"r"))==NULL){
        fprintf(stderr,"error₩n");
```

```
        }
        while(fgets(buffer,sizeof(buffer),fp)!=NULL){
                n= strlen(buffer);
                count++;
        }
        m=count;
        printf("m = %d n = %d₩n",m,n);
        fclose(fp);
        char **temp_matrix;
        temp_matrix = (char **)malloc(sizeof(char *)*m);
        for(int i =0;i <m;i ++){
                temp_matrix[i] = (char *)malloc(sizeof(char)*n);
        }
        /*while(fgets(buffer,sizeof(buffer),fp)!=NULL){
                buffer[strlen(buffer)-1] ='₩0';
                strcpy(matrix[index],buffer);
                index++;
        }
        reset_matrix(m,n,temp_matrix,matrix);*/
        while(1){
                int thread_num;
                int input_num;
                int gen_num;
                int process_num;
                printf("input 1.exit 2. normal  3. Process  4. Thread : ");
                scanf("%d",&input_num);
                if(input_num ==1){
                        printf("thank you .₩n");
                        break;
                }
                else if(input_num ==2){
                        clock_t start, end;
                        int index =0;
                        double result;
                        printf("please input generation number :
");scanf("%d",&gen_num);
                        start = clock();
                        for(int i =0;i <gen_num;i ++){
                                char filename[100];
                                char ex_filename[100];
        sprintf(filename,"gen_%d.matrix",i +1);
                                if(i ==0)
                                        strcpy(ex_filename,argv[1]);
                                else
                                        sprintf(ex_filename,"gen_%d.matrix",i);
                                if(i +1 ==gen_num)
                                        strcpy(filename,"output.txt");
        FILE *temp_fp;
                                if((temp_fp = fopen(ex_filename,"r"))==NULL){
                                        fprintf(stderr,"error open file₩n");
                                        exit(1);
                                }
                                while(fgets(buffer,sizeof(buffer),temp_fp)!=NULL){
                                        if(buffer[strlen(buffer)-1] =='₩n')
```

```
                                        buffer[strlen(buffer)-1] ='₩0';
                                strcpy(temp_matrix[index],buffer);
                                //printf("%s₩n",temp_matrix[index]);
                                index++;
                        }
                        index =0;
                memset(buffer,0,sizeof(buffer));
                        fclose(temp_fp);
                        fp = fopen(filename,"w+");
                        fclose(fp);
                        game(m,n,temp_matrix,0,m,filename);
                }
                        end = clock();
                        result = (double)(end - start);
                        printf("func time : %fms₩n",result);
                }
                else if(input_num ==3 ){
                        clock_t start_t,end_t;
                        double result;
                        int start =0;
                        int jump;
                        int index =0;
                        char buffer_matrix[m][n];
                        printf("input the number of child process : ");
                        scanf("%d",&process_num);
                        int count_process = process_num;
                        int count_m = m;
                        printf("please input generation number :
");scanf("%d",&gen_num);
                        pid_t child_pid[gen_num][process_num], wpid;
                        int status =0;
                        start_t = clock();
                        for(int i =0;i <gen_num;i ++){
                                char filename[100];
                char ex_filename[100];
                sprintf(filename,"gen_%d.matrix",i +1);
                if(i ==0)
                    strcpy(ex_filename,argv[1]);
                else
                    sprintf(ex_filename,"gen_%d.matrix",i);
                if(i +1 ==gen_num)
                    strcpy(filename,"output.txt");
                FILE *temp_fp;
                if((temp_fp = fopen(ex_filename,"r"))==NULL){
                    fprintf(stderr,"error open file₩n");
                    exit(1);
                }
                while(fgets(buffer,sizeof(buffer),temp_fp)!=NULL){
                    if(buffer[strlen(buffer)-1] =='₩n')
                        buffer[strlen(buffer)-1] ='₩0';
                    strcpy(temp_matrix[index],buffer);
                    //printf("%s₩n",temp_matrix[index]);
                    index++;
                }
```

```
                index =0;
                memset(buffer,0,sizeof(buffer));
                fclose(temp_fp);
                fp = fopen(filename,"w+");
                fclose(fp);
                                for (int id =0;id <process_num;id ++){
                                        if((child_pid[i][id] = fork())==0){
                                                jump = count_m / count_process;
                                                count_m -= jump;
                                                count_process -=1;

game(m,n,temp_matrix,start,start+jump,filename);
                                                start+=jump;
                                                exit(0);
                                        }
                                        else if(child_pid[i][id] >0){
                                                jump = count_m /count_process;
                                                count_m -=jump;
                                                count_process-=1;
                                                start+=jump;
                                        }
                                }
                                start =0;
                                jump =0;
                                count_m =m;
                                count_process = process_num;
                                while((wpid = wait(&status))>0);
                        }
                        end_t = clock();
                        result = (double)(end_t - start_t);
                        for(int i =0;i <gen_num;i ++){
                                printf("%dst gen₩n",i +1);
                                for(int j =0;j <process_num;j ++){
                                        printf("pid %d₩n",child_pid[i][j]);
                                }
                                printf("₩n");
                        }
                        printf("func time : %fms₩n",result);
                }

                else if (input_num ==4){
                        clock_t start_t, end_t;
                        int start =0;
                        int jump =0;
                        /*THREAD st;
                        st.matrix = (char **)malloc(sizeof(char*) * m);
                        for(int i =0;i<m;i++){
                                st.matrix[i] = (char *)malloc(sizeof(char) *n);
                        }
                        st.filename = (char *)malloc(sizeof(char*)*100);*/
        int index =0;
        double result;
                        printf("please input number of thread :
");scanf("%d",&thread_num);
```

```
printf("please input generation number : ");scanf("%d",&gen_num);
                int count_m = m;
                int count_thread = thread_num;
                THREAD* st;
                st = (THREAD *)malloc(sizeof(THREAD) * thread_num);

for(int k =0;k <thread_num;k ++){
                        st[k].matrix = (char **)malloc(sizeof(char *) * m);
  for(int i =0;i <m;i ++){
     st[k].matrix[i] = (char *)malloc(sizeof(char) *n);
  }
  st[k].filename = (char *)malloc(sizeof(char *)*100);
                }
                pthread_t tid[thread_num];
start_t = clock();
for(int i =0;i <gen_num;i ++){
    char filename[100];
    char ex_filename[100];
    sprintf(filename,"gen_%d.matrix",i +1);
    if(i ==0)
        strcpy(ex_filename,argv[1]);
    else
        sprintf(ex_filename,"gen_%d.matrix",i);
    if(i +1 ==gen_num)
        strcpy(filename,"output.txt");
    FILE *temp_fp;
    if((temp_fp = fopen(ex_filename,"r"))==NULL){
        fprintf(stderr,"error open file₩n");
        exit(1);
    }
    while(fgets(buffer,sizeof(buffer),temp_fp)!=NULL){
        if(buffer[strlen(buffer)-1] =='₩n')
            buffer[strlen(buffer)-1] ='₩0';
        strcpy(temp_matrix[index],buffer);
                                //strcpy(st.matrix[index],buffer);
        index++;
    }
    index =0;
    memset(buffer,0,sizeof(buffer));
    fclose(temp_fp);
    fp = fopen(filename,"w+");
    fclose(fp);
                        int err;
                        printf("₩n%dst gen₩n",i +1);
                        for(int j =0;j <thread_num;j ++){
                                st[j].m = m;
                                st[j].n = n;
                                //st[j].matrix = (char **)malloc(sizeof(
                                st[j].matrix = temp_matrix;
                                st[j].start = start;
                                jump = count_m / count_thread;
                                count_m -=jump;
                                count_thread -=1;
                                st[j].end = start + jump;
```

```
                                        strcpy(st[j].filename,filename);
                                        start+=jump;
                                        //printf("* m : %d n : %d start : %d end :
%d filename : %s₩n",st[j].m,st[j].n,st[j].start,st[j].end,st[j].filename);
                                        if(err                                          =
pthread_create(&tid[j],NULL,thread_func,(void *)&st[j])){
                                                fprintf(stderr,"error in thread₩n");
                                                exit(1);
                                        }
                                //      pthread_detach(tid[j]);
                                }
                                start =0;
                                jump =0;
                                count_m = m;
                                count_thread = thread_num;
                                for(int j =0;j <thread_num;j ++)
                                        pthread_join(tid[j],NULL);
        }
                        printf("₩n");
        end_t = clock();
        result = (double)(end_t - start_t);
        printf("func time : %fms₩n",result);

                }
                else {



                        printf("please input 1~4 number₩n");
                }
        }
        return 0;
}
```

3. 실행 결과

```
home@ubuntu:~/HW_Lyn$ ./project input2.matrix
m = 1000 n = 1999
input 1.exit 2. tnsck 3. Process  4. Thread : 2
please input generation number : 2
func time : 58060.000000ms
input 1.exit 2. tnsck 3. Process  4. Thread : 3
input the number of child process : 5
please input generation number : 2
1st gen
pid 3549
pid 3550
pid 3551
pid 3552
pid 3553

2st gen
pid 3554
pid 3555
pid 3556
pid 3557
pid 3558

func time : 5156.000000ms
input 1.exit 2. tnsck 3. Process  4. Thread : 4
please input number of thread : 5
please input generation number : 2

1st gen
140481624127232 140481657698048 140481632519936 140481640912640 140481649305344
2st gen
140481657698048 140481649305344 140481640912640 140481624127232 140481632519936
func time : 57058.000000ms
input 1.exit 2. tnsck 3. Process  4. Thread :
```

1000 * 1000 행렬일 때 수행시간 비교
각 5개의 프로세스, 5개의 쓰레드를 비교했다.
순차처리 (normal) 의 경우는 58000 ms ,
프로세스 병렬처리 의 경우 5156 ms
스레드 병렬처리의 경우 57000 ms

여러 번 시도 해본 결과 대부분의 결과는
Process > Thread >=normal 의 속도로 측정 되었다.

각 세대 별로 사용한 process의 id 와 thread의 id를 출력하였다.

ex) 2개의 process를 가지고 4개의 generation을 거친다면
2 *4 = 8, 사실상 총 8개의 프로세스를 사용한 것과 같다.

4. 구현을 통해 배운점(아쉬운 점)

(0) 스레드 병렬처리 시 gcc -pthread
크게 중요한 내용은 아니기 때문에 0번으로 설정하였다. 멀티 쓰레드를 위해 <pthread.h> 헤더를 포함시켰고 pthread_create0 등의 함수를 정상적으로 리눅스 내에서 실행시키기 위해서는 gcc에 옵션으로 -pthread를 포함시켜야한다.
ex) gcc -o project.c project.c -pthread

(1) vfork 와 fork 의 차이점

처음 멀티 프로세스 병렬처리를 구현 할 때 메모리를 공유하기 위해 사용했던 함수는 vfork 였다. 구현이 끝난 후 처리시간이 좀처럼 순차처리보다 빠르지 못했고 이를 해결하기 위해 공부해보니 vfork 는 부모프로세서의 메모리를 사용하기 위해 여러 개의 자식프로세서가 동시에 참조할 수 없고 이는 곧 각 자식프로세서들이 동시에 진행되는 것이 아닌 각각을 기다리게 되는 것과 처리 시간이 비슷해 질 거 라는 생각이 들었다.

이후 fork()를 사용하여 고쳤고 수행시간 문제는 해결 할 수 있었다. vfork 와 fork 의 차이점은 부모 프로세서의 메모리를 복사한는 것 만이 아닌 사용하고 수정할수 있느냐 이고 여기서 생기는 차이점은 수행시간의 차이를 불러온다는 것을 배웠다.

(2) 병렬처리와 순차처리의 수행시간 비교

대용량의 파일을 다루게 되면 역시 병렬 처리가 빠를 수 밖에 없다. 수행될 데이터 양이 커지면 커질수록 그 차이는 더욱 커질 것이다. 하지만 데이터 양이 작을 때는 아래의 사진과 같이 순차 처리가 더욱 빠르다. 왜 그런 것일까.

```
input 1.exit 2. tnsck 3. Process  4. Thread : 4
please input number of thread : 3
please input generation number : 2

1st gen
140509519623936 140509511231232 140509502838528
2st gen
140509502838528 140509519623936 140509511231232
func time : 785.000000ms
input 1.exit 2. tnsck 3. Process  4. Thread : 3
input the number of child process : 3
please input generation number : 2
1st gen
pid 3635
pid 3636
pid 3637

2st gen
pid 3638
pid 3639
pid 3640

func time : 855.000000ms
input 1.exit 2. tnsck 3. Process  4. Thread : 2
please input generation number : 2
func time : 202.000000ms
input 1.exit 2. tnsck 3. Process  4. Thread :
```

※ 각각 202ms , 855ms , 785ms 가 걸렸다.

병렬처리에서는 프로세서를 만들고 메모리를 복사하고 , 스레드를 만들고 각 실행 시간을 할당하는 등의 다른 작업들이 필요하다 이것들이 오버헤드라고 부를 수 있는데 실제 함수의 수행 시간보다 오버헤드 실행시간이 더 긴 환경에서는 당연히 순차처리가 제일 작은 실행 시간을 갖게 될 것이다.

* 아쉬운 점 : 멀티 스레드 구현 오류?

멀티 스레드를 활용한 병렬처리를 구현하였지만 수행 시간이 납득 가지 않는다. 프로세서의 경우 수행시간의 감소를 뚜렷히 파악할 수 있는데 멀티 스레드에서는 순차처리와 같거나 오히려 순차처리보다 느린 경우도 종종 나왔다. 각 컴퓨터 환경이나 다른 요소들이 작용한 것도 있겠지만 왜 이렇게 나오는 것인지 정확히 모르겠다.

여러 곳에서 알아본 결과 스레드가 루프 문에서 굉장히 느리다 라던지 함수 실행 시간 측정 함수가 잘못 됐다던 지라는 이유라고 파악하였다...

내가 생각했을 때의 수행시간이 길게 걸리는 이유는 pthread_join 때문이 아닐 까 싶다. main thread는 모든 자식 thread를 기다리게 해야 하는 데 반복문 안에서 이를 진행하였고, 첫 번째 스레드의 종료가 되었을 때 다음 스레드의 종료를 확인한다. 이는 모든 자식 스레드 중에서 실행 시간이 제일 빨라서 종료된 스레드 혹인 주어진 데이터 양이 적어 먼저 종료된 스레드 등을 파악하는 게 아니라 pthread_create()를 한 순서대로 끝났는지를 확인하는 것이 되어 순차처리와 전체 시간이 비슷해 진 게 아닌가 싶다. 함수 수행시간 자체는 적은데 기다리는 시간이 모두 포함되어 버린 듯 하다.

결론적으로 3가지 방식을 통한 세포 생명 게임을 구현하였다.