シミュレーション ～宿題１:M/M/1 モデル～

e055717 金城佑典

2008/12/09

目次

1 M/M/1 model

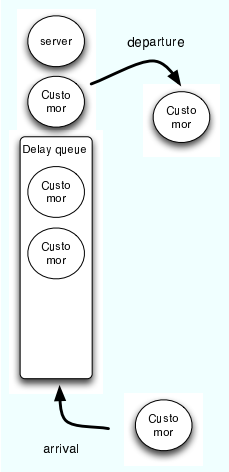

図 1 M/M/1 model

M/M/1 モデルとはケンドールの記法（要求の到着頻度/処理時間分布/窓口の数）で表記された待ち行列で以下の 3 つの条件を満たすもののことである。

1. 到着間隔がランダム（ポアゾン分布に従う）
2. 処理時間がランダム（指数分布に従う）
3. サービス窓口は 1 個

このとき「割り込みはなし」「到着順にサービスを受ける」という条件があり、例えば銀行の窓口業務やプリンタの待ち行列などが M/M/1 モデルにあたる。評価基準は主に以下の 3 つである。

1. 平均待ち人数 (Avarage number of customer in queue)
2. 平均待ち時間 (Average delay in queue)
3. 稼働率 (Server utilization)

2 シミュレーションプログラム

2.1 ヘッダファイル（mm1.h）

2.1.1 ソースコード

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

#include <getopt.h>//実行時オプションの判定
#include <time.h>//random 関数のシード値を決めるのに使う
#include <stdbool.h>//bool 型を使う
```

```
#define Q_LIMIT 100 //queue の上限

/*server state*/
#define BUSY 1
#define IDLE 0

int next_event_type,//次 の イ ベ ン ト の 種 類 、1->Arrival(到 着) 2->Departure(出
発) (3->End(終了) (Limit Time Mode))
    num_custs_delayed,//待たされた客の数
    num_deleyed_required,//客の上限 (Limit Costomer Number mode)
    num_events,//イベント数
    num_in_q,//待っている客の数
    server_status;//店員の状態 (BUSY or IDLE)

int arrival_id=0,//到着した客の ID
    dep_id=0;//出発した客の人数

float   area_num_in_q,//のべ待ち時間
        area_server_status,//店員ののべ仕事時間
        mean_interarrival,//次の客が来るまでの時間
        mean_service,//対応にかかる時間
        sim_clock,//時計
        time_end,//終了時間 (Fixed Time mode)
        time_arrival[Q_LIMIT+1],//到着時間
        time_last_event,//最終イベント時間
        time_next_event[4],//次のイベントの時間、1->出発時間 2->到着時間 (3->終了時
間 (Limit Time Mode) )
        total_of_delays;//総待ち時間

FILE    *infile,//入力ファイル (mm1.in)
        *outfile;//出力ファイル (mm1.out)

/*functions*/

void initialize();
void timing(void);
void arrival(void);
void depart(void);
void report(void);
```

```
void update_time_avg_status(void);
float expon(float mean);

void usage(char *myname);
```

3　シミュレーションプログラム

3.1　mian (Simulation body)

3.1.1　説明

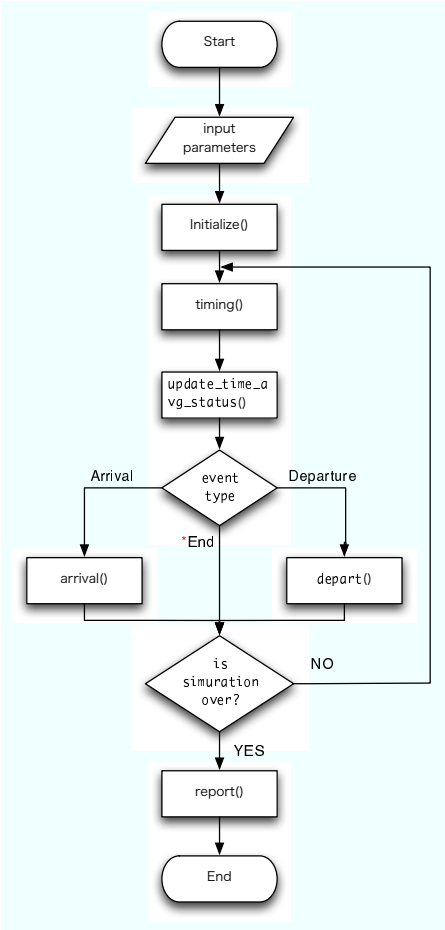

図 2　Simulation flow

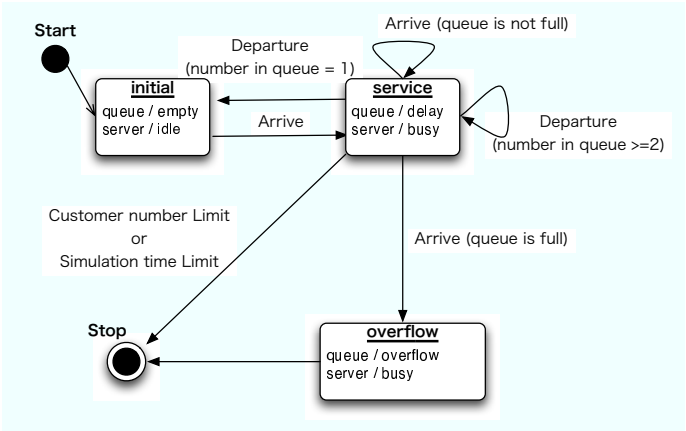

図 3　状態遷移図

まず、ファイルからパラメータを読み込みシミュレーションを初期化する。その後シミュレーションを開始するる。ここでシミュレーションの終了条件は「指定した人数の客が来た」と「営業時間が終了した」のどちらかを選べるようにした。(これらはシミュレー

ション開始時に実行オプションで選択できるようにした。）

シミュレーションループではまず timing 関数で次のイベントを決定する、ここで生成するイベントは「客の到着 (Arrival)」と「客の出発 (Depearture)」の 2 つ（「営業時間が終了した」を終了条件にする場合は「営業終了 (Simulation End)」を含めた 3 つ）になる。

次に update_time_avg_status 関数でのべ時間などを計算した後、イベントを発生させる。

シミュレーションが終了条件に達したらレポートを出力してプログラムを終了する。

3.1.2 ソースコード

```
int main(int argc, char **argv){

    char *infilename = "mm1.in";
    char *outfilename = "mm1.out";

    //Read Exec options（実行オプションの読み込み）
    int option;
    while( (option = getopt(argc, argv,"i:o:th"))!=-1 ){
        switch(option){
            case 'i':
                infilename = optarg;
                break;
            case 'o':
                outfilename = optarg;
                break;
            case 't':
                limit_time_mode = true;
                break;
            case 'h':
            default:
                usage(argv[0]);
                return 0;
                break;
        }
    }

    /*Open input and output file（入力ファイルと出力ファイルを開く）*/
    infile  =   fopen(infilename,"r");
```

```
    outfile  =   fopen(outfilename,"w");

    /*Specify the number of events for the timing function（イベント数を定義）*/
    //Customer Number Limit Mode
    if(!limit_time_mode) num_events = 2;//到着(Arrival Event)と出発(Departure Event)
    //Time Limit Mode
    else num_events = 3;//到着 (Arrival Event) と出発 (Departure  Event) と終了
(Simulation End  Event)

    /*Read input parameter.（パラメータの読み込み）*/
    if(!limit_time_mode) {//Customer Number Limit Mode
        fscanf(infile,"%f %f %d",&mean_interarrival,&mean_service,&num_deleyed_required);
        printf("input %f %f %d\n",mean_interarrival,mean_service,num_deleyed_required);
    }else{//Time Limit Mode
        fscanf(infile,"%f %f %f",&mean_interarrival,&mean_service,&time_end);
        printf("input %f %f %f\n",mean_interarrival,mean_service,time_end);
    }

    if(!limit_time_mode) printf("Exec Limit Costomer Number Mode\n");
    else printf("Exec Limit Time Mode\n");

    /*Write Report heading and input parameter.（入力されたパラメータを書き出し）*/
    fprintf(outfile,"Single-server queueing system \n\n");
    fprintf(outfile,"Mean interarrival time  %11.3f minutes \n",mean_interarrival);
    fprintf(outfile,"Mean service time %16.3f minutes \n",mean_service);

    if(!limit_time_mode)
        fprintf(outfile,"Number of customers %14d \n",num_deleyed_required);
    else fprintf(outfile,"Length of the Simulation %9.3f minute \n",time_end);


    printf("mean interarrival %f \n",mean_interarrival);
    printf("mean service %f \n",mean_service);

    /*Initialize the simulation.（シミュレーションの初期化）*/
    initialize();

    bool sim_run=true;
    while(sim_run){
        /*Determine the next event.（次のイベントを定義）*/
```

```
        timing();

        /*Update time-average statistical accumulators（のべ時間の更新）*/
        update_time_avg_status();

        /*Invoke the appropriate event function.（イベントの実行）*/
        switch(next_event_type){
            case 1:
                arrival_id++;
                printf("[%10.3f] Customer %d is Arrival\n",sim_clock,arrival_id);
                arrival();//Arrival Event(到着イベント)
                break;
            case 2:
                dep_id++;
                printf("[%10.3f] Customer %d is Departure\n",sim_clock,dep_id);
                depart();//Departure Event(出発イベント)
                break;
            case 3://for Limit Time Mode
                /*Invoke the report generator and end the simulation.（レポート
を出力してシミュレーションを修了）*/
                report();
                break;
        }
        if(!limit_time_mode){//Limit Customer Number Mode
            /*Quit the simulation when more delays are not needed.（もう客が来な
くてもいいならシミュレーションを終わる）*/
            if(num_custs_delayed > num_deleyed_required) sim_run=false;
        }else{//Limit Time Mode
            /*Quit the simulation when time to simulation end.（シミュレーション
の終了時間になったらシミュレーションを終わる）*/
            if(next_event_type==3) sim_run=false;
        }
    }

    /*Invoke the report generator and end the simulation.（レポートを出力してシミ
ュレーションを修了）*/
    if(!limit_time_mode) report();//Limit Customer Number Mode

    fclose(infile);
    fclose(outfile);
```

```
    return 0;
}
```

3.2 initialize (Initialize simulation)

3.2.1 説明

シミュレーションの初期化を行う関数。expon については次節に譲る。

Event2(Departure イベント) の初期時間を timing 関数で定義されている min_time_next_event よりも大きくして、Arrival イベントが先に発生するようにしているところに注意。

また、シミュレーションの終了条件を時間にする場合は Event3(Simuration end イベント) の時間を設定する。

3.2.2 ソースコード

```
/*Initialization function（初期化）*/
void initialize(void){
/*Set rondom seed(ランダムのシード値の設定)*/
    srand((unsigned int)time(NULL));

    /*Initialize the Simulation sim_clock（シミュレーション内の時計の初期化）*/
    sim_clock = 0.0f;

    /*Initialize the state variables（状態の初期化）*/
    server_status   =   IDLE;
    num_in_q        =   0;
    time_last_event =   0.0f;

    /*Initialize the statical counters（カウンタの初期化）*/
    num_custs_delayed   =   0;
    total_of_delays     =   0.0f;
    area_num_in_q       =   0.0f;
    area_server_status  =   0.0f;

    /*Initialize event list.（イベントリストの初期化）
        Since no customers are present,（客が存在しないなら）
        the departure (service completion) event is eliminated from consideration.
```

```
        (「departure」イベントは無視する）*/
    time_next_event[1]  =   sim_clock + expon(mean_interarrival);
    /*Guaranteening that first event will be an arrival
    (最初のイベントを Arrival にするために、([timing()]min_time_next_event より）大きな数を設定しておく)*/
    time_next_event[2]  =   1.0e+30;
    /*if Limit Time Mode, set Time to Simulation end*/
    if(limit_time_mode) time_next_event[3] = time_end;
}
```

3.3 expon (Exponential variate generation)

3.3.1 説明

mean(β) を基にランダムな Exponential を計算する関数。（イベント発生時間のランダム化はここで行っている。）

$$\int_0^x \frac{1}{\beta} e^{-\frac{1}{\beta}} dt$$
$$= 1 - e^{-\frac{1}{\beta}}$$

ここで Probability 関数は

$$P(-\beta \ln U >= x)$$
$$= P(\ln U <= -\frac{1}{\beta})$$
$$= P(U <= e^{-\frac{1}{\beta}})$$
$$= P(e^{-\frac{1}{\beta}} <= U <= 1)$$
$$= 1 - e^{-\frac{1}{\beta}}$$

なので、Exponential の計算には $-\beta \ln U$ を用いる

3.3.2 ソースコード

```
/*Exponential variate genaration function（指数の変量を計算）*/
float expon(float mean){
    float u;

    /*Generate a U(0,1) random variate（0<=u<=1 の乱数）*/
    u = (float)rand()/RAND_MAX;
```

```
    /*Return an exponential random variate with mean "mean"
(mean を基に計算した Exponential Variate を返す）*/
    return -mean * log(u);

}
```

3.4 timing (Next event setting)

3.4.1 説明

次のイベントを決定し、シミュレーションの時計を進める関数。
initialize 関数の Depearture イベントの初期値はここで定義されている min_time_next_event の値よりも大きくなければならない、そうすることによって Arrival イベントより先に Depearture イベントが発生しないようにする。

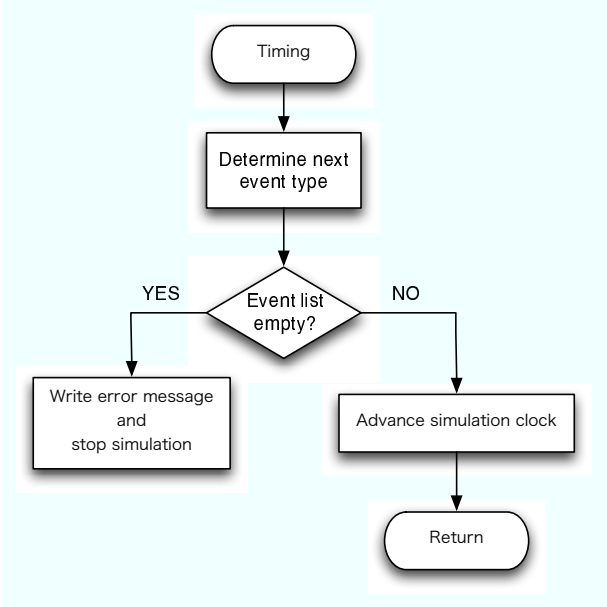

図 4　timing 関数のフローチャート

3.4.2 ソースコード

3.4.3 ソースコード

```
/*Timing function*/
void timing(void){
    int i;
    float min_time_next_event = 1.0e+29;

    next_event_type = 0;

    /*Determine the event type of next event occur(次に発生するイベント生成する)*/
    for(i=1; i <= num_events; ++i){
```

```
        if(time_next_event[i] < min_time_next_event){
            min_time_next_event = time_next_event[i];
            next_event_type = i;
        }
    }

    /*Check to see whether the event list is empty（イベントリストが空でないかチェックする）*/
    if(next_event_type == 0){
        /*the event list is empty, so stop simulation（イベントリストが空なのでシミュレーション停止）*/
        fprintf(outfile,"\nEvent list empty at sim_clock %f",sim_clock);
        exit(1);
    }

    /*The event list is not empty, so advance the simulation sim_clock
（イベントリストが空でないので、時計を進める）*/
    sim_clock = min_time_next_event;
}
```

3.5 update_time_avg_state (Update area accumulator)

3.5.1 説明

平均時間を計算するために、「シミュレーション時間」と「のべ待ち時間」と「のべ作業時間」を計算する。

3.5.2 ソースコード

```
/*Update area acumulator for time-average（のべ時間の更新）*/
void update_time_avg_status(void){
    float sim_clock_since_last_event;

    /*Compute sim_clock since last event,（最終イベントからの経過時間を計算し）
        and update last-event-sim_clock marker.（最終イベント時間を更新）*/
    sim_clock_since_last_event = sim_clock - time_last_event;
    time_last_event = sim_clock;

    /*Update area under number-in-queue function.（のべ待ち時間更新）*/
    area_num_in_q += num_in_q * sim_clock_since_last_event;
```

```
    /*Update area under server-busy indicator function.（のべ作業時間の更新）*/
    area_server_status += server_status * sim_clock_since_last_event;
}
```

3.6 arrive (Arrival event)

3.6.1 説明

Arrival イベントの関数。
次の Arrival イベントの時間を設定した後、Server が忙しいなら Customer を待たせて（待ち行列に入れて）到着時間 (Arrival time) を記憶する。（待ち行列があふれたらシミュレーションを中止する。)Server が暇ならサービスを開始し、Server の状態を BUSY にしてから Department イベントをスケジュールする。

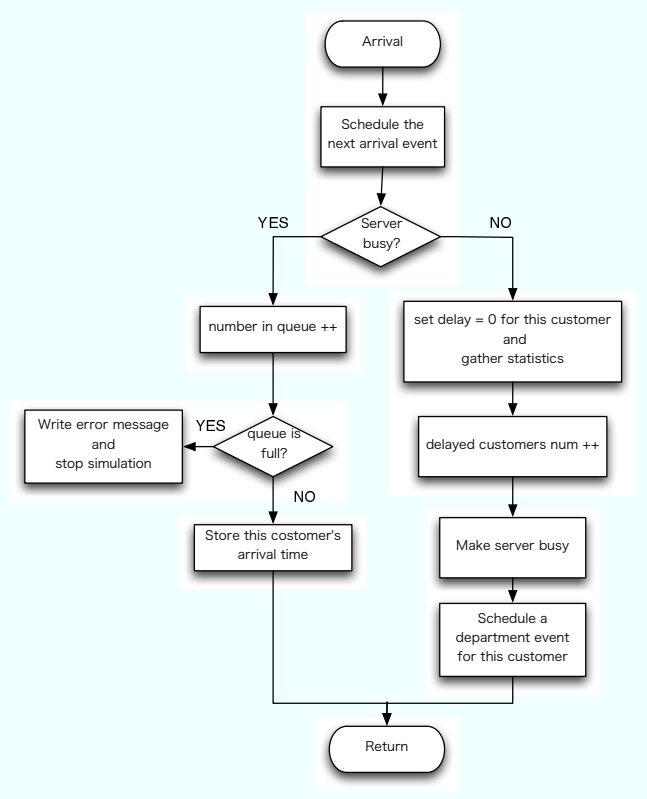

図 5　Arrival 関数のフローチャート

3.6.2 ソースコード

```
/*Arrival event function（到着イベント）*/
void arrival(void){
    float delay;

    /*Schedule next arrival（次の到着をスケジュールに追加）*/
    time_next_event[1] = sim_clock + expon(mean_interarrival);
```

```
    /*Check the whether server is busy（店員が忙しいかどうかチェック）*/
    if(server_status == BUSY){
        /*server is busy, so increment number of customers in queue
（店員が忙しいので、待ち行列内の客の数を増やす）*/
        ++num_in_q;

        /*Check to see whether an overflow condition status
（客があふれていないかチェック）*/
        if(num_in_q > Q_LIMIT){
            /*the queue has overflowed, so stop the simulation
（客があふれているのでシミュレーション停止）*/
            fprintf(outfile,
    "\nOverflow of the array time_arrival at sim_clock %f",sim_clock);
            exit(2);
        }

        /*there is still room in the queue,（まだ待ち行列内にいるので）
            so store the sim_clock of arrival of the arriving costomer
            at the (new) end of time_arrival
             （time_arrival に客の到着時間を追加する）*/
        time_arrival[num_in_q] = sim_clock;

    }else{
        /*server is idle,（店員はヒマ）
            so arriving customer has delay of zero.（なので待ち時間はなし）
            the following two statement are for program clarity
               and do not affect the result of the simulation
             （一応明記しておくが、シミュレーションの結果には影響がない）*/
        delay = 0.0;
        total_of_delays += delay;

        /*Increment the number of customers delayed,（待っていた客の数を増やし）
            and make server busy（店員を仕事中にする）*/
        ++num_custs_delayed;
        server_status = BUSY;

        /*Schedule a dedeparture(service completion)
（出発イベントをスケジュールに追加）*/
        time_next_event[2] = sim_clock + expon(mean_service);
    }
```

```
}
```

3.7 depart (Departure event)

3.7.1 **説明**

Departure イベントの関数。
待ち行列が空なら、Server を IDLE にして出発イベントが呼ばれないようにする。待っている客がいたら、待ち行列から一人呼び出して、その客の待ち時間を計算し、待っていた客の数を加算した後、この客の Departure イベントをスケジュールしする。

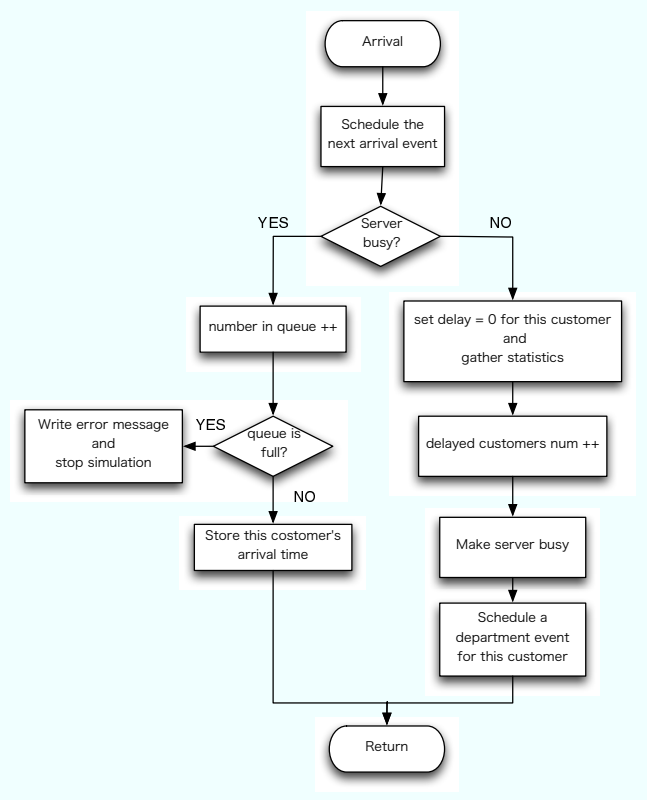

図 6　depart 関数のフローチャート

3.7.2 **ソースコード**

```
/*Departure event function（出発イベント）*/
void depart(void){
    int i;
    float delay;

    /*Check to see whether the queue is empty（客が待っているか調べる）*/
    if(num_in_q == 0){
        /*queue is empty so make server idle
      and eliminate the departure event from consideration
         （待ってる人がいないので店員をヒマにし、出発イベントを無視する）*/
```

```
        server_status = IDLE;
        //min_time_next_event より確実に大きいので timing で発生されなくなる
        time_next_event[2]= 1.0e+30;
    }else{
        /*queue is nonempty,（だれかが待っている）
            so decrement the number of customers in queue.（行列から一人減らす）*/
        --num_in_q;

        /*Compute the delay of customer who is beginning service
        and update the total delay accumulator
        （その人が待っていた時間を計算し、総待ち時間を増やす）*/
        delay = sim_clock - time_arrival[1];
        total_of_delays += delay;

        /*Increment the number of costomers delayed,（待っていた客の数を増やし）
            and schedule departure.（出発イベントをスケジュールに追加する）*/
        ++num_custs_delayed;
        time_next_event[2] = sim_clock + expon(mean_service);

        /*Move each customer in queue (if any) up one place.
（待ち行列から客を削除し前につめる）*/
        for(i=1; i <= num_in_q; ++i) time_arrival[i] = time_arrival[i+1];
    }
}
```

3.8 report (output report)

3.8.1 説明

シミュレーションの実行結果を表示する。

3.8.2 ソースコード

```
/*Report generator function（結果表示）*/
void report(void){
    /*Compute and write estimates of desired mesures of perfomance.*/
    fprintf(outfile,"\nAvarage delay in queue %11.3f minutes",total_of_delays/num_custs_dela
平均待ち時間
    fprintf(outfile,"\nAvarage number in queue %10.3f",area_num_in_q/sim_clock);//
平均待ち人数
```

```
    fprintf(outfile,"\nServer Utilization %15.3f",area_server_status/sim_clock);//
店員の仕事率
    fprintf(outfile,"\ntime simulation ended %12.3f",sim_clock);//シミュレーショ
ン時間
    fprintf(outfile,"\nVisited %d customers",arrival_id);//来店客数
    fprintf(outfile,"\nServiced %d customers",dep_id);//来店客数
}
```

3.9 usage (print usage)

3.9.1 説明

実行時オプションの表示を行う。

3.9.2 ソースコード

```
void usage(char *myname){
    printf("%s [hdt][-i Input_File_Name][-o Output_File_Name]\n",myname);
    printf("\t -t:Limit Time Mode (default:Limit Costomer Number Mode)\n");
    printf("\t -i <Input_File_Name>:set Input File (default:mm1.in)\n");
    printf("\t -o <output_File_Name>:set Output File (default:mm1.out)\n");
    printf("\t -h:show this usage\n");
    printf("\n");
    printf("Input File's format (Limit Costomer Number Mode)\n--- \n");
    printf("[(float)mean interva llimit[minute]] [(float)mean service limit[minute]] [(int)d
    printf("ex)1.000 0.5000 1000\n");
    printf("\n");
    printf("Input File's format (Limit Time Mode)\n--- \n");
    printf("[(float)mean interva llimit[minute]] [(float)mean service limit[minute]] [(float
    printf("ex)1.000 0.5000 100\n");
}
```

4 シミュレーションの実行

シミュレーションの実行には前述した通り２通りのモードがある。

どちらの場合も、以下のような入力ファイルを用意する

```
1.000 0.5000 1000
```

左の 2 つは順に「(float) Mean Interval [minute]」と「(float) Mean Service [minute]」である。3 つ目は Limit Customer Mode の時は「(int) Number Deleyed Required [peaple]」で Limit Time Mode の際は「(float) Simulation End Time [minute]」である。

4.1 Limit Customer Mode (n=1000)

客の人数を制限するモード、「Mean Interval」を 1 分、「Mean Service」を 0.5 分 (30 秒)、「Number Deleyed Required」(n) を 1000 人で実行すると、以下のような結果が得られた。

```
[~/Documents/4-2/Simulation/MM1]%./mm1
input 1.000000 0.500000 1000
Exec Limit Costomer Number Mode
mean interarrival 1.000000
mean service 0.500000
[    11.758] Customer 1 is Arrival
[    11.898] Customer 1 is Departure
[    13.786] Customer 2 is Arrival
[    14.101] Customer 2 is Departure
[    14.566] Customer 3 is Arrival
.
.
.
```

実行結果ファイルの中身は

```
Single-server queueing system
Mean interarrival time       1.000 minutes
Mean service time            0.500 minutes
Number of customers           1000

Avarage delay in queue       0.441 minutes
Avarage number in queue      0.436
Server Utilization           0.487
time simulation ended     1013.619
Visited 1002 customers
Serviced 1000 customers
```

つまり、n=1000 でシミュレーションを実行すると平均待ち時間 0.441 分、平均待ち人数は 0.436 人、稼働率は 48.7 パーセントだったことがわかる。来客数とサービスを受け

られた人数より、2 人はサービスを受けられずに帰った事になる。

4.2 Limit Time Mode

シミュレーションの実行時間を制限するモード、「Mean Interval」を 1 分、「Mean Service」を 0.5 分 (30 秒)、「Simulation End Time」を 1000 分で実行すると、以下のような結果が得られた。

```
[e055717:~/Documents/4-2/Simulation/MM1]%./mm1 -t -o out.data
input 1.000000 0.500000 1000.000000
Exec Limit Time Mode
mean interarrival 1.000000
mean service 0.500000
[    11.758] Customer 1 is Arrival
[    11.898] Customer 1 is Departure
[    13.786] Customer 2 is Arrival
[    14.101] Customer 2 is Departure
[    14.566] Customer 3 is Arrival
.
.
.
```

実行結果ファイルの中身は

```
Single-server queueing system
Mean interarrival time       1.000 minutes
Mean service time            0.500 minutes
Length of the Simulation  1000.000 minute

Avarage delay in queue       0.414 minutes
Avarage number in queue      0.408
Server Utilization           0.480
time simulation ended     1000.000
Visited 985 customers
Serviced 983 customers
```

Mean interarrival と Mean service time が同じなので、客数を限定したときとだいたいおなじ結果になっている。

参考文献

[1] SIMULATION MODELING & ANALYSIS : Averill M. Law , W David Kelton

[2] [ITpro] M/M/1
http://itpro.nikkeibp.co.jp/article/COLUMN/20060920/248528/?ST=start2