MMSK

程式碼步驟

```
struct node{
    double arrivaltime;
    double servicetime;
    double departime;
    struct node *l;
};
```

將每個來的封包由link list 存放arrivaltime, servicetime和departime

初始化方式

```
struct Node *creat(double arrivaltime , double servicetime){
    struct node *temp=(struct node *)malloc(sizeof(struct node));
    temp->arrivaltime= arrivaltime;
    temp->servicetime= servicetime;
    temp->departime= arrivaltime;
    temp->l=NULL;
    return temp;
}
```

其中封包之間的arrivaltime計算式參考自

> https://github.com/lulululuej/MMCK_simulation

```
double poisson_intertime(double rate){

    double interarrivaltime= -log(lcgrand())* (60.0/rate);
    return interarrivaltime;

}
```

lcgrand() 為引入函式庫

以下為封包根據poisson分布得到的arrivaltime與servicetime

```
struct node *getarrivalpacketinfo(double lambda, double mu, int n){
    double tempservicetime = poisson_intertime(mu);
    struct node *root = creat(0, tempservicetime);
    struct node *temp=root;
    int i;
    for(i=0;i<n-1;i++){
        double tempintertime = poisson_intertime(lambda);
        tempservicetime = poisson_intertime(mu);
        temp->l= creat(temp->arrivaltime + tempintertime, tempservicetime);
        temp=temp->l;
    }
    return root;
}
```

再來對應每個server結構設置

是否被使用

以及離去時間

```
struct hd{
    int serv_used;
    double departime;
};
```

主要計算方式

選取每個對應server中離去時間的最小值

若來的封包時間較小

則判斷進入arrival

若來的封包時間較大

則開始進行server中封包離開系統的時間計算

```
while(temp != NULL){
            if(temp->arrivaltime < find_min_depart(Serv, S)){
                arrive(Serv, temp, S , K, mu, Q);
                temp=temp->l;
            }
            else{
                departure(Serv, S, K, mu, Q);
            }
        }
```

arrive

```
void arrive(struct hd Serv[], struct node *temp, int S , int K, double mu, struct Queue Q[]){
    int j;
    for(j=0;j<S;j++){
        if(Serv[j].serv_used == 0){
            temp->departime=temp->arrivaltime + temp->servicetime;
            Serv[j].departime = temp->departime;
            Serv[j].serv_used = 1;
            return;
        }
    }
    if(num_Q <= K-1){
        Q[num_Q].l=temp;
        num_Q++;
    }
    return;
}
```

當有server是空閒的進入server並計算離去時間

沒有的話就進入Queue

若是Queue已滿則直接丟掉

departure

```
void departure(struct hd Serv[], int S , int K, double mu, struct Queue Q[]){
    int i;
    for(i=0;i<S;i++){
        if(find_min_depart(Serv, S) == Serv[i].departime){
            break;
        }
    }
    if(num_Q == 0){
        Serv[i].serv_used = 0;
        Serv[i].departime = INT_MAX;
    }
    else{
        num_Q--;
        Q->l->departime = Serv[i].departime + Q->l->servicetime;
        Serv[i].departime = Q->l->departime;
        for(i=0;i<num_Q+1;i++){
            Q[i].l=Q[i+1].l;
            Q[i+1].l=NULL;
        }
        return;
    }
}
```

找尋Queue中最早進入的點取代server中最小值

並計算離去時間

若是Queue為空則設置server為idle狀態

最終計算方式

```
while(temp->l!= NULL){
            if(temp->departime != temp->arrivaltime){
                W+=temp->departime - temp->arrivaltime;
                totalservicetime+=temp->servicetime;
                count++;
            }
            temp=temp->l;
        }
        printf("%.8lf, ", W/(temp->departime)); //L
        printf("%.8lf, ", (W-totalservicetime)/(temp->departime)); //Lq
        printf("%.8lf, ", W/count/60); //W
        printf("%.8lf\n", (W-totalservicetime)/count/60); //Wq
```

L==(全部封包的departime-arrivaltime)/系統總時間

Lq==((全部封包的departime-arrivaltime) -系統總服務時間)/系統總時間

W==(全部封包的departime-arrivaltime)/封包數—-這邊/60是單位換算

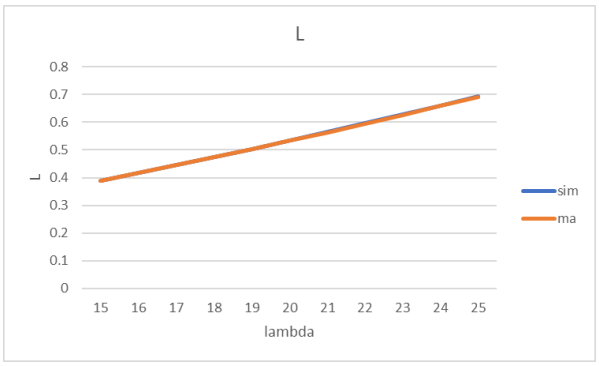
Wq==((全部封包的departime-arrivaltime) -系統總服務時間)/封包數—-這邊/60是單位換算

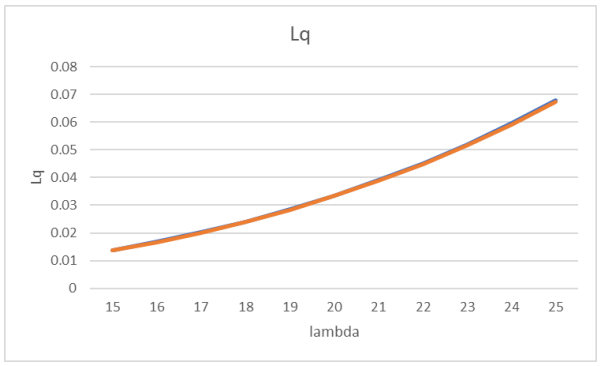
模擬與數學對比圖

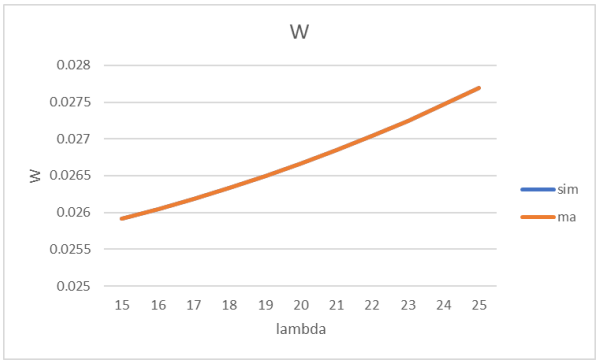
數學模擬與程式模擬差距在0.1%左右

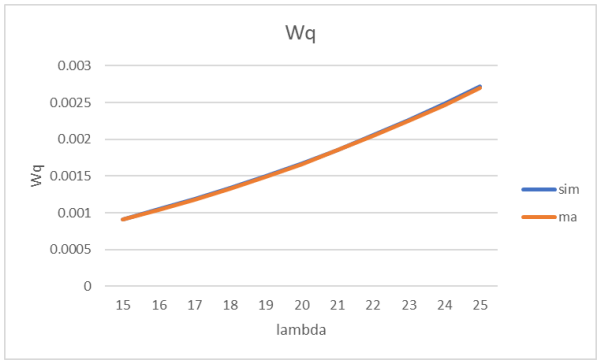
參數設置

```
int S = 2, K = 8, n=1000000;
double lambda=25.0;
double mu=40.0;
```


L
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
L
15 16 17 18 19 20 21 22 23 24 25
lambda
sim
ma


Lq
0.08
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0
Lq
15 16 17 18 19 20 21 22 23 24 25
lambda


W
0.028
0.0275
0.027
0.0265
0.026
0.0255
0.025
W
15 16 17 18 19 20 21 22 23 24 25
lambda
sim
ma


Wq
0.003
0.0025
0.002
0.0015
0.001
0.0005
0
Wq
15 16 17 18 19 20 21 22 23 24 25
lambda
sim
ma