/\*優先度関数値が小さい順にソートする関数\*/

void LMCLF(){

double priority\_func[S]; //優先度関数値格納変数（作業用）

double memory=0,minmemory=MAX; //メモリ消費量格納変数

double priority\_func1=0,priority\_func2=0;

int i = 0, j = 0, k = 0,l = 0; //カウント用変数

double alphauppermin=MAX,alphalowermax=0; //αの範囲最大最小

double alphaupperminsav=MAX,alphalowermaxsav=0; //αの範囲最大最小(保存用)

double tempalpha1=0,tempalpha2=0; //仮のαの上限下限格納関数

double WCETLaxity1=0,WCETLaxity2=0; //残余実行時間×余裕時間

double randma1=0,randma2=0; //α×消費メモリ増分

int keta1=0,keta2=0; //α×消費メモリ増分と残余実行時間×余裕時間の桁数を引いたもの

int besti,bestk; // 最小メモリとなるiとkを記憶

pthread\_mutex\_lock(&mutex);

alphadiff=0;

/\*換算レートαの決定\*/

for(i=0;i<TN;i++){

alphauppermin=MAX,alphalowermax=0;

if(state[i] == 1){

for(j=0;j<TN;j++){

if(state[j] == 1){

　if(i==j){

}else if(rand\_memory[i][step[i]] < rand\_memory[j][step[j]]){

// m(i)α+Ci\*Li<m(j)α+Cj\*Lj && m(j)>m(i) --> Ci\*Li-Cj\*Lj<(m(j)-m(i))α --> α>(Ci\*Li-Cj\*Lj)/(m(j)-m(i))

tempalpha1=(((task\_data[i].WCET - step[i]) \* task\_data[i].Laxity\_Time)-((task\_data[j].WCET - step[j]) \*

task\_data[j].Laxity\_Time))/((rand\_memory[j][step[j]])-(rand\_memory[i][step[i]]));

WCETLaxity1=(((task\_data[i].WCET - step[i]) \* task\_data[i].Laxity\_Time) + ((task\_data[j].WCET - step[j]) \*

task\_data[j].Laxity\_Time))/2;

randma1=(fabs(tempalpha1)\*(fabs(rand\_memory[i][step[i]]) + fabs(rand\_memory[j][step[j]])))/2;

if(alphalowermax<tempalpha1){

alphalowermax=tempalpha1;

keta1=get\_digit(randma1) - get\_digit(WCETLaxity1);

}else{

}

}else{ // m(i)α+Ci\*Li<m(j)α+Cj\*Lj && m(j)<m(i) --> (m(i)-m(j))α<Cj\*Lj-Ci\*Li --> α<(Cj\*Lj-Ci\*Li)/(m(i)-m(j))

tempalpha2=(((task\_data[j].WCET - step[j]) \* task\_data[j].Laxity\_Time)-((task\_data[i].WCET - step[i]) \*

task\_data[i].Laxity\_Time))/((rand\_memory[i][step[i]])-(rand\_memory[j][step[j]]));

WCETLaxity1=(((task\_data[j].WCET - step[j]) \* task\_data[j].Laxity\_Time) + ((task\_data[i].WCET - step[i]) \*

task\_data[i].Laxity\_Time))/2;

randma1=(fabs(tempalpha2)\*(fabs(rand\_memory[i][step[i]]) + fabs(rand\_memory[j][step[j]])))/2;

if(alphauppermin>tempalpha2){

alphauppermin=tempalpha2;

keta1=get\_digit(randma1) - get\_digit(WCETLaxity1);

}

}

}

}

if(!(alphauppermin>0 && alphalowermax < alphauppermin)){

continue;

}

fprintf(stderr,"%lf <= alpha <= %lf for scheduling task %d first\n",alphalowermax,alphauppermin,i+1);

alphaupperminsav=alphauppermin; alphalowermaxsav=alphalowermax;

for(k=0;k<TN;k++){

alphauppermin=alphaupperminsav; alphalowermax=alphalowermaxsav;

if(state[k] == 1){

for(l=0;l<TN;l++){

if(state[l] == 1){

if(k==l){

}else if(rand\_memory[k][(k==i)?(step[k]+1):step[k]] < rand\_memory[l][(l==i)?(step[l]+1):step[l]] ){

// m(k)α+Ck\*Lk<m(l)α+Cl\*Ll && m(l)>m(k) --> Ck\*Lk-Cl\*Ll<(m(l)-m(k))α --> α>(Ck\*Lk-Cl\*Ll)/(m(l)-m(k))

tempalpha1=(((task\_data[k].WCET - (k==i)?(step[k]+1):step[k]) \* task\_data[k].Laxity\_Time)-((task\_data[l].WCET –

(l==i)?(step[l]+1):step[l])\*task\_data[l].Laxity\_Time))/((rand\_memory[l][(l==i)?(step[l]+1):step[l]])-

(rand\_memory[k][(k==i)?(step[k]+1):step[k]]));

WCETLaxity2=(((task\_data[k].WCET - step[k]) \* task\_data[k].Laxity\_Time) + ((task\_data[l].WCET - step[l]) \*

task\_data[l].Laxity\_Time))/2;

randma2=(fabs(tempalpha1)\*(fabs(rand\_memory[k][step[k]]) + fabs(tempalpha1)\*fabs(rand\_memory[l][step[l]])))/2;

if(alphalowermax<tempalpha1){

alphalowermax=tempalpha1;

keta2=get\_digit(randma2) - get\_digit(WCETLaxity2);

}else{

}

}else{

// m(k)α+Ck\*Lk<m(l)α+Cl\*Ll && m(l)<m(k) --> (m(k)-m(l))α<Cl\*Ll-Ck\*Lk --> α<(Cl\*Ll-Ck\*Lk)/(m(k)-m(l))

tempalpha2=(((task\_data[l].WCET - (l==i)?(step[l]+1):step[l]) \* task\_data[l].Laxity\_Time)-((task\_data[k].WCET

-(k==i)?(step[k]+1):step[k])\*task\_data[k].Laxity\_Time))/((rand\_memory[k][(k==i)?

(step[k]+1):step[k]])-(rand\_memory[l][(l==i)?(step[l]+1):step[l]]));

WCETLaxity2=(((task\_data[l].WCET - step[l]) \* task\_data[l].Laxity\_Time) + ((task\_data[k].WCET - step[k]) \*

task\_data[k].Laxity\_Time))/2;

randma2=(fabs(tempalpha2)\*(fabs(rand\_memory[l][step[l]])+fabs(tempalpha2)\*

fabs(rand\_memory[k][step[k]])))/2;

if(alphauppermin>tempalpha2){

alphauppermin=tempalpha2;

keta2=get\_digit(randma2) - get\_digit(WCETLaxity2);

}

}

}

}

if(alphauppermin>0 && alphalowermax < alphauppermin){

fprintf(stderr,"%lf <= alpha <= %lf for scheduling task %d and then task %d\n",alphalowermax,alphauppermin,i+1,k+1);

if(i==k){

if(rand\_memory[i][step[i]+1]>0){

memory=rand\_memory[i][step[i]] + rand\_memory[i][step[i]+1];

}else{

memory=rand\_memory[i][step[i]];

}

}else{

if(rand\_memory[k][step[k]]>0){

memory=rand\_memory[i][step[i]] + rand\_memory[k][step[k]];

}else{

memory=rand\_memory[i][step[i]];

}

}

if(minmemory>memory){

minmemory=memory; besti=i; bestk=k;

if(alphauppermin<MAX){

if(((keta1+keta2)/2)>0){

alpha=((alphalowermax + alphauppermin)/2)/(pow(10,(keta1+keta2)/2));

}else{

alpha=((alphalowermax + alphauppermin)/2);

}

}else{

if(((keta1+keta2)/2)>0){

alpha=(alphalowermax)/(pow(10,(keta1+keta2)/2));

}else{

alpha=(alphalowermax);

}

}

}

}

}

}

}

}