

COL331

1 Distributed Algorithm: How exactly does your algorithm work?

First, I have calculated the total sum and variance using two different for loops in a distributed algorithm, the first one is an explanation of total sum part.

```
int par_id=getpid();

for(int i=0;i<n_procs;i++){
    int ch_id=fork();
    if(ch_id==0){
        int sum=0;
        for(int j=i*(size/n_procs);j<(i+1)*(size/n_procs);j++){
            sum+=arr[j];
        }
        // printf(1,"partial sum %d\n", sum);
        for(int j=0;j<10000;j++){
            send(getpid(),par_id,6,sum);
            exit();
        }
    }
    int rec_sum=0;
    for(int i=0;i<n_procs;i++){
        while(recv(&rec_sum)==-1){}
        // printf(1,"receive sum %d\n", rec_sum);
        tot_sum+=rec_sum;
    }
}
```

Figure 1: Distributed algorithm:Total sum

In , this parid is used to store the coordinator process which will accumulate the sum created by the child processes. The for loop is used in which fork is used which creates new child process. So, here if the chid is 0 then the particular process is a child process. So, we use this knowledge that this process should be used to calculate partial sum. In this we divide the 1000 size array into 125 size chunks and each child process works on itself given 125 size chunk. The for loop after that is used to provide a little wait. Then send uses the pid of the given process and send the sum to the specified parent whose id is stored in parid. After the loop completes all the send shall reach to parent process and then using the loop it sums all the partial sum received using sys receive system call to total sum.

```

66 float var=0.0;
67 int ch_id=0;
68 int child_id[n_procs];
69 par_id=getpid();
70 int flag,cnt=0;
71 for(int i=0;i<n_procs;i++)
72 {
73     ch_id=fork();
74     if(ch_id==0){
75         flag=0;
76         child_id[i]=ch_id;
77     }
78     else{
79         flag=1;
80         break;
81     }
82     cnt++;
83 }
84 if(flag==1)
85 {
86     float a=0;
87     while(recv(&a)==-1){};
88     float sum=0;
89     for(int j=cnt*(size/n_procs);j<(cnt+1)*(size/n_procs);j++){
90         sum+=(arr[j]-a)*(arr[j]-a);
91     }
92     send(getpid(),par_id,&sum);
93     exit();
94 }
95 else{
96     send_multi(par_id,child_id,&mean);
97     float rec_ans=0;
98     for(int i=0;i<n_procs;i++){
99         while(recv(&rec_ans)==-1){};
100         // printf("partial sq-sum %d\n", (int)(rec_ans*1000));
101         var+=rec_ans/size;
102     }
103 }
104 }
105

```

Figure 2: Distributed algorithm: Variance

After calculating the total sum, this code calculates the variance. the parid again used to store the parent id. The for loop creates the different child processes of parent processes. In the loop if the process is a parent process then the array childid is used to store all the child process ids to this process. If it is a child process then loop that process comes out of loop as we have to compute the variance this child process is used to calculate the the different element in a chunk subtracted with the mean which is sent by the parent process to different child processes using send multi system call, then this calculated answer is sent to parent process then parent process receives this answer from all the child processes then calculates the variance by summing them and dividing them by the size of array.

2 IPC: Explain the implementation of the inter-rupt handler

In this part three functions are made one for send, one for receive and one for send multi.

```

#include "spinlock.h"

typedef struct{
    int ptr_read;
    int ptr_write;
}process_no;

process_no pro = {
    .ptr_read=0,
    .ptr_write=0
};

process_no run_proc[NPROC];
struct spinlock lock;
char arr_buff[NPROC][512];

```

Figure 3: Struct used in send and receive

This new structure is created whose objects are created in the functions created.

```

int sys_send(int sender_pid,int rec_pid,void *msg)
{
    char* ch;
    argint(0,&sender_pid);
    argint(1,&rec_pid);
    argptr(2,&ch,8);
    acquire(&lock);
    if((run_proc[rec_pid].ptr_write+8)%512==run_proc[rec_pid].ptr_read){
        release(&lock);
        return -1;
    }
    else{
        int i=0;
        while(i<8){
            arr_buff[rec_pid][run_proc[rec_pid].ptr_write]= *(ch+i);
            run_proc[rec_pid].ptr_write=(run_proc[rec_pid].ptr_write+1)%512;
            i++;
        }
        msg=ch;
        // printf("%s\n",arr_buff[rec_pid][write_pid]);
        // printf("This is the message %s\n,msg);
        // printf("This is the message %s\n",*ch);
        release(&lock);
        // printf("send is successful %s");
        return 0;
    }
}

```

Figure 4: This function is created to implement send

This function is created in which argint and argptr are used to take input of sender pid , recive pid and the message. In this the basic algorithm used is a type of circular buffer in this for each pid a buffer is created whose size is 512 , this number is selected as it is multiple of 8 which is the size of message received. Here two pointers are used which are defined in the structure are read pointer and write pointer which are used to point to the point till which we have written in a respective pid buffer. If the message written and it gets overlap then we say that we cant write any more in the buffer and return -1 in that case. Here locks are used using spinlock which are used to lock a particular thread which are writing to the particular buffer.

```

}

// code for send receive
int sys_rcv(void* msg)
{
    char* ch;
    if (argptr(0, &ch, 0) < 0)
    {
        return -1;
    }
    int curr_id = myproc()->pid;
    // if(read_pid[curr_id])=write_pid[curr_id])
    // printf("This is the message\n",ch);
    acquire(&lock);
    if(run_proc[curr_id].ptr_read==run_proc[curr_id].ptr_write){
        release(&lock);
        return -1;
    }
    else{
        for(int i=0;i<8;i++){
            *(ch+i)=arr_buff[curr_id][run_proc[curr_id].ptr_read];
            run_proc[curr_id].ptr_read=(run_proc[curr_id].ptr_read+1)%512;
        }
    }
    release(&lock);
    return 0;
}

int sys_send_multi(int sender_pid, int rec_pids[], void *msg){

```

Figure 5: This function is created to implement receive

This function input is same as earlier. Here we receive then read the particular message from the buffer.

```

int sys_send_multi(int sender_pid, int rec_pids[], void *msg){
    char* ch;
    int* arr;
    argint(0, &sender_pid);
    argptr(1, (char**) &arr, 64);
    argptr(2, &ch, 0);
    acquire(&lock);
    for(int i=0;i<8;i++){
        if(arr[i]<0){
            release(&lock);
            return -1;
        }
        if((run_proc[arr[i]].ptr_write%512==run_proc[arr[i]].ptr_read){
            printf("No more writes can be made here %s",ch);
            release(&lock);
            return -1;
        }
        else{
            int j=0;
            while(j<8){
                arr_buff[arr[i]][run_proc[arr[i]].ptr_write]= *(ch+j);
                run_proc[arr[i]].ptr_write=(run_proc[arr[i]].ptr_write+1)%512;
                j++;
            }
        }
        msg=ch;
    }
    release(&lock);
    return 0;
}

```

Figure 6: This function is created to implement send multi

This function is same as send just here that an array of fixed size 8 is given, for loop is used and same type of code structure used in send function is used in this for loop.

3 Print Count

```

14 char* arr1["yes sir",
15 "yes little",
16 "yes class",
17 "yes mom",
18 "yes aunt",
19 "yes uncle",
20 "yes baby",
21 "yes teacher",
22 "yes friend",
23 "yes waiter",
24 "yes little",
25 "yes lady",
26 "yes mother",
27 "yes son",
28 "yes sister",
29 "yes prince count",
30 "yes pal",
31 "yes queen",
32 "yes king",
33 "yes prince",
34 "yes doctor",
35 "yes uncle",
36 "yes used multi",
37 "yes class",
38 "yes teacher",
39 "yes little",
40 "yes queen",
41 "yes mother",
42 "yes uncle",
43 "yes wife"];
44
45 get
46 {
47     cout<<endl;
48
49     char* arr1["yes sir",
50 "yes little",
51 "yes class",
52 "yes mom",
53 "yes aunt",
54 "yes uncle",
55 "yes baby",
56 "yes teacher",
57 "yes friend",
58 "yes waiter",
59 "yes little",
60 "yes lady",
61 "yes mother",
62 "yes son",
63 "yes sister",
64 "yes prince count",
65 "yes pal",
66 "yes queen",
67 "yes king",
68 "yes prince",
69 "yes doctor",
70 "yes uncle",
71 "yes used multi",
72 "yes class",
73 "yes teacher",
74 "yes little",
75 "yes queen",
76 "yes mother",
77 "yes uncle",
78 "yes wife"];
79
80     int n=sizeof(arr1)/sizeof(char*);
81
82     for(int i=0;i<n;i++)
83     {
84         cout<<arr1[i]<<" ";
85     }
86
87     cout<<endl;
88
89     int n1=sizeof(arr1)/sizeof(char*);
90
91     for(int i=0;i<n1;i++)
92     {
93         cout<<arr1[i]<<" ";
94     }
95
96     cout<<endl;
97
98     int n2=sizeof(arr1)/sizeof(char*);
99
100    for(int i=0;i<n2;i++)
101    {
102        cout<<arr1[i]<<" ";
103    }
104
105    cout<<endl;
106
107    int n3=sizeof(arr1)/sizeof(char*);
108
109    for(int i=0;i<n3;i++)
110    {
111        cout<<arr1[i]<<" ";
112    }
113
114    cout<<endl;
115
116    int n4=sizeof(arr1)/sizeof(char*);
117
118    for(int i=0;i<n4;i++)
119    {
120        cout<<arr1[i]<<" ";
121    }
122
123    cout<<endl;
124
125    int n5=sizeof(arr1)/sizeof(char*);
126
127    for(int i=0;i<n5;i++)
128    {
129        cout<<arr1[i]<<" ";
130    }
131
132    cout<<endl;
133
134    int n6=sizeof(arr1)/sizeof(char*);
135
136    for(int i=0;i<n6;i++)
137    {
138        cout<<arr1[i]<<" ";
139    }
140
141    cout<<endl;
142
143    int n7=sizeof(arr1)/sizeof(char*);
144
145    for(int i=0;i<n7;i++)
146    {
147        cout<<arr1[i]<<" ";
148    }
149
150    cout<<endl;
151
152    int n8=sizeof(arr1)/sizeof(char*);
153
154    for(int i=0;i<n8;i++)
155    {
156        cout<<arr1[i]<<" ";
157    }
158
159    cout<<endl;
160
161    int n9=sizeof(arr1)/sizeof(char*);
162
163    for(int i=0;i<n9;i++)
164    {
165        cout<<arr1[i]<<" ";
166    }
167
168    cout<<endl;
169
170    int n10=sizeof(arr1)/sizeof(char*);
171
172    for(int i=0;i<n10;i++)
173    {
174        cout<<arr1[i]<<" ";
175    }
176
177    cout<<endl;
178
179    int n11=sizeof(arr1)/sizeof(char*);
180
181    for(int i=0;i<n11;i++)
182    {
183        cout<<arr1[i]<<" ";
184    }
185
186    cout<<endl;
187
188    int n12=sizeof(arr1)/sizeof(char*);
189
190    for(int i=0;i<n12;i++)
191    {
192        cout<<arr1[i]<<" ";
193    }
194
195    cout<<endl;
196
197    int n13=sizeof(arr1)/sizeof(char*);
198
199    for(int i=0;i<n13;i++)
200    {
201        cout<<arr1[i]<<" ";
202    }
203
204    cout<<endl;
205
206    int n14=sizeof(arr1)/sizeof(char*);
207
208    for(int i=0;i<n14;i++)
209    {
210        cout<<arr1[i]<<" ";
211    }
212
213    cout<<endl;
214
215    int n15=sizeof(arr1)/sizeof(char*);
216
217    for(int i=0;i<n15;i++)
218    {
219        cout<<arr1[i]<<" ";
220    }
221
222    cout<<endl;
223
224    int n16=sizeof(arr1)/sizeof(char*);
225
226    for(int i=0;i<n16;i++)
227    {
228        cout<<arr1[i]<<" ";
229    }
230
231    cout<<endl;
232
233    int n17=sizeof(arr1)/sizeof(char*);
234
235    for(int i=0;i<n17;i++)
236    {
237        cout<<arr1[i]<<" ";
238    }
239
240    cout<<endl;
241
242    int n18=sizeof(arr1)/sizeof(char*);
243
244    for(int i=0;i<n18;i++)
245    {
246        cout<<arr1[i]<<" ";
247    }
248
249    cout<<endl;
250
251    int n19=sizeof(arr1)/sizeof(char*);
252
253    for(int i=0;i<n19;i++)
254    {
255        cout<<arr1[i]<<" ";
256    }
257
258    cout<<endl;
259
260    int n20=sizeof(arr1)/sizeof(char*);
261
262    for(int i=0;i<n20;i++)
263    {
264        cout<<arr1[i]<<" ";
265    }
266
267    cout<<endl;
268
269    int n21=sizeof(arr1)/sizeof(char*);
270
271    for(int i=0;i<n21;i++)
272    {
273        cout<<arr1[i]<<" ";
274    }
275
276    cout<<endl;
277
278    int n22=sizeof(arr1)/sizeof(char*);
279
280    for(int i=0;i<n22;i++)
281    {
282        cout<<arr1[i]<<" ";
283    }
284
285    cout<<endl;
286
287    int n23=sizeof(arr1)/sizeof(char*);
288
289    for(int i=0;i<n23;i++)
290    {
291        cout<<arr1[i]<<" ";
292    }
293
294    cout<<endl;
295
296    int n24=sizeof(arr1)/sizeof(char*);
297
298    for(int i=0;i<n24;i++)
299    {
300        cout<<arr1[i]<<" ";
301    }
302
303    cout<<endl;
304
305    int n25=sizeof(arr1)/sizeof(char*);
306
307    for(int i=0;i<n25;i++)
308    {
309        cout<<arr1[i]<<" ";
310    }
311
312    cout<<endl;
313
314    int n26=sizeof(arr1)/sizeof(char*);
315
316    for(int i=0;i<n26;i++)
317    {
318        cout<<arr1[i]<<" ";
319    }
320
321    cout<<endl;
322
323    int n27=sizeof(arr1)/sizeof(char*);
324
325    for(int i=0;i<n27;i++)
326    {
327        cout<<arr1[i]<<" ";
328    }
329
330    cout<<endl;
331
332    int n28=sizeof(arr1)/sizeof(char*);
333
334    for(int i=0;i<n28;i++)
335    {
336        cout<<arr1[i]<<" ";
337    }
338
339    cout<<endl;
340
341    int n29=sizeof(arr1)/sizeof(char*);
342
343    for(int i=0;i<n29;i++)
344    {
345        cout<<arr1[i]<<" ";
346    }
347
348    cout<<endl;
349
350    int n30=sizeof(arr1)/sizeof(char*);
351
352    for(int i=0;i<n30;i++)
353    {
354        cout<<arr1[i]<<" ";
355    }
356
357    cout<<endl;
358
359    int n31=sizeof(arr1)/sizeof(char*);
360
361    for(int i=0;i<n31;i++)
362    {
363        cout<<arr1[i]<<" ";
364    }
365
366    cout<<endl;
367
368    int n32=sizeof(arr1)/sizeof(char*);
369
370    for(int i=0;i<n32;i++)
371    {
372        cout<<arr1[i]<<" ";
373    }
374
375    cout<<endl;
376
377    int n33=sizeof(arr1)/sizeof(char*);
378
379    for(int i=0;i<n33;i++)
380    {
381        cout<<arr1[i]<<" ";
382    }
383
384    cout<<endl;
385
386    int n34=sizeof(arr1)/sizeof(char*);
387
388    for(int i=0;i<n34;i++)
389    {
390        cout<<arr1[i]<<" ";
391    }
392
393    cout<<endl;
394
395    int n35=sizeof(arr1)/sizeof(char*);
396
397    for(int i=0;i<n35;i++)
398    {
399        cout<<arr1[i]<<" ";
400    }
401
402    cout<<endl;
403
404    int n36=sizeof(arr1)/sizeof(char*);
405
406    for(int i=0;i<n36;i++)
407    {
408        cout<<arr1[i]<<" ";
409    }
410
411    cout<<endl;
412
413    int n37=sizeof(arr1)/sizeof(char*);
414
415    for(int i=0;i<n37;i++)
416    {
417        cout<<arr1[i]<<" ";
418    }
419
420    cout<<endl;
421
422    int n38=sizeof(arr1)/sizeof(char*);
423
424    for(int i=0;i<n38;i++)
425    {
426        cout<<arr1[i]<<" ";
427    }
428
429    cout<<endl;
430
431    int n39=sizeof(arr1)/sizeof(char*);
432
433    for(int i=0;i<n39;i++)
434    {
435        cout<<arr1[i]<<" ";
436    }
437
438    cout<<endl;
439
440    int n40=sizeof(arr1)/sizeof(char*);
441
442    for(int i=0;i<n40;i++)
443    {
444        cout<<arr1[i]<<" ";
445    }
446

```

Figure 7: This function is created to implement print count

[illegible]

Figure 8: This function is created to implement print count

This system call is used to print the system call processes running in alphabetical order. The sorting is used in this is like an array is defined which contains all the system calls in a sorted order. Then all are compared to all the elements and then putted in a new array and then printed.

4 sysps

```

3 void process_status(void);
4 int
5 sys_ps(void)
6 {
7     process_status();
8     return 0;
9 }
10

```

Figure 9: This function is created to implement ps system call

```

535
536 void
537 process_status(void)
538 {
539     struct proc *p;
540
541     acquire(&table.lock);
542
543     for(p = table.proc; p < &table.proc[NPROC]; p++){
544         if (p->state != UNUSED) {
545             cprintf("pid:%d name:%s\n", p->pid, p->name);
546         }
547     }
548     release(&table.lock);
549 }

```

Figure 10: This function is created to implement ps system call

In this the process status keeps the status of the of all the processes which is used by the ps system call.

This is the screenshot of the output of check.sh bash.

```

411+1 records in
411+1 records out
218994 bytes (211 kB, 286 KiB) copied, 0.0011346
Running..1
Running..2
Running..3
Running..4
Running..5
Running..6
Running..7
Running..8 (this will take 10 seconds)
Test #1: PASS
Test #2: PASS
Test #3: PASS
Test #4: PASS
Test #5: PASS
Test #6: PASS
Test #7: PASS
Test #8: PASS
8 test cases passed
varnish77@varnish77:~/xv6-public-2$

```

Figure 11: Test cases Result