

OS Assignment:-

STUDENT NAME : - RONIT MEHTA

STUDENT ID : - 11803363

EMAIL ADDRESS: - ronitmehta853@gmail.com

GIT HUB LINK : - <https://github.com/ronit-lab/os-316assignment>

TO see code :- <https://github.com/ronit-lab/os-316assignment/blob/master/.gitignore>

COURSE TITLE: - OPERATING SYSTEM

COURSE CODE :- CSE-316

PROBLEM :-

In the given problem create a Process Id (PID) manager that help us to know about free PIDs and it also check

that no two processes are having the same pid value Once a process terminates the PID manager may provide its pid to new process.

Use the following variables or constants to know the range of possible pid values:

```
#define MIN PID 100
```

```
#define MAX PID 1000
```

You may use any data structure of your know to represent process identifiers.

You can use the following functions :-

- `int allocate map(void)`—Creates and initializes a any data structure for which can use to represent pids;

returns—1 if fail, 1 if sucess

- `int allocate pid(void)`—provide pid to each unique process and returns a pid; returns— 1 which means cannot allocate else return 1 can allocate

a pid (all pids are in use)

- `void release pid(int pid)`—Releases a pid

Modify the given problem by making a multithreaded program that examine your solution.

You will create a number of threads—for example, 100—and each thread will allocate with a pid,which is unique

sleep for a random period of time, and then release the pid .The release pid is assign to new process.

SOLUTION :-

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

```
#include<stdlib.h>

#include<time.h>

#include<unistd.h>

#include<pthread.h>

#include<sys/types.h>

#define MIN_PID 100

#define MAX_PID 1000
```

```
int bit_map[MAX_PID-MIN_PID]={0};
```

```
int allocate_map()
```

```
{
```

```
for(int i=0; i< MAX_PID;i++)
```

```
bit_map[i]=0;
```

```
}
```

```
int allocate_pid(void){
```

```
int i,flag=1;
```

```
for(i=0; i<MAX_PID-MIN_PID; i++){
```

```
if(bit_map[i]==0){
```

```
bit_map[i]=1;
```

```
flag=0;
```

```

        break;
    }
}
return flag?-1:i;
}
void release_pid(int id){
    bit_map[id]=0;
}
void *threading(void *az){
    int tid = *((int*)az);

    int id = allocate_pid();

    if(id==-1){
        printf("No PID available.");
    }
    else{
        printf("Thread  %d is allocated with PID %d \n",tid,id+MIN_PID);

        sleep(10);
    }
}

```

```

    printf("Thread (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);

    release_pid(id);

}

    printf("released PID is assign to new thread %d\n");
    pthread_exit(NULL);

}

void *processin(void *az){
    int tid = *((int*)az);

    int id = allocate_pid();

    if(id==-1){
        printf("No PID available.");
    }
    else{
        printf("Process %d is allocated with PID %d \n",tid,id+MIN_PID);

        sleep(10);
    }
}

```

```
printf("Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);

release_pid(id);


}

printf("released PID is assign to new process %d\n");


pthread_exit(NULL);


}


int main(){

    printf(">>>>>>>>>>>>>>>>>>>WELCOME TO PID
MANAGER>>>>>>>>>>>\n");

    allocate_pid();


    int num;

    printf("<<<<<<<<<<<<<<<<<<<<<ENTER YOUR
CHOICE>>>>>>>>>>>>>>>\n");

    printf("1) FOR PROCESS CREATION \n");

    printf("2) FOR THREAD CREATION \n");


    scanf("%d",&num);

    switch(num)

    {

        case 1:

            printf("<<ITS PROCESS CREATION SECTION\n");
```

```

        int a;

        int tid;

        int id;

    int NO_OF_Processes;

    printf("enter the no of processes \n");

    scanf("%d",&NO_OF_Processes);

    for(int i=0;i<NO_OF_Processes;i++)
    {

        pthread_t process[NO_OF_Processes];

        for(i=0; i<NO_OF_Processes; i++){
        if(pthread_create(&process[i],NULL,processin,(void*)&i))
            return -1*printf("Error in process %d creation !!!\n",i);
        }

        for(i=0; i<NO_OF_Processes; i++)
            pthread_join(process[i],NULL);

        }

        break;

```

case 2:

```
printf("<<ITS THREAD CREATION SECTION MAIN TESTCASE PRESS  
100 \n");
```

```
int i;
```

```
int NO_OF_THREADS;
```

```
printf("enter the no of threads \n");
```

```
scanf("%d",&NO_OF_THREADS);
```

```
pthread_t process[NO_OF_THREADS];
```

```
for(i=0; i<NO_OF_THREADS; i++){
```

```
    if(pthread_create(&process[i],NULL,threading,(void*)&i))
```

```
        return -1*printf("Error in thread %d creation !!!\n",i);
```

```
}
```

```
for(i=0; i<NO_OF_Processes; i++)
```

```
    pthread_join(process[i],NULL);
```

```
return 0*printf("\n*****PID DONE ITS WORK*****\n");
```

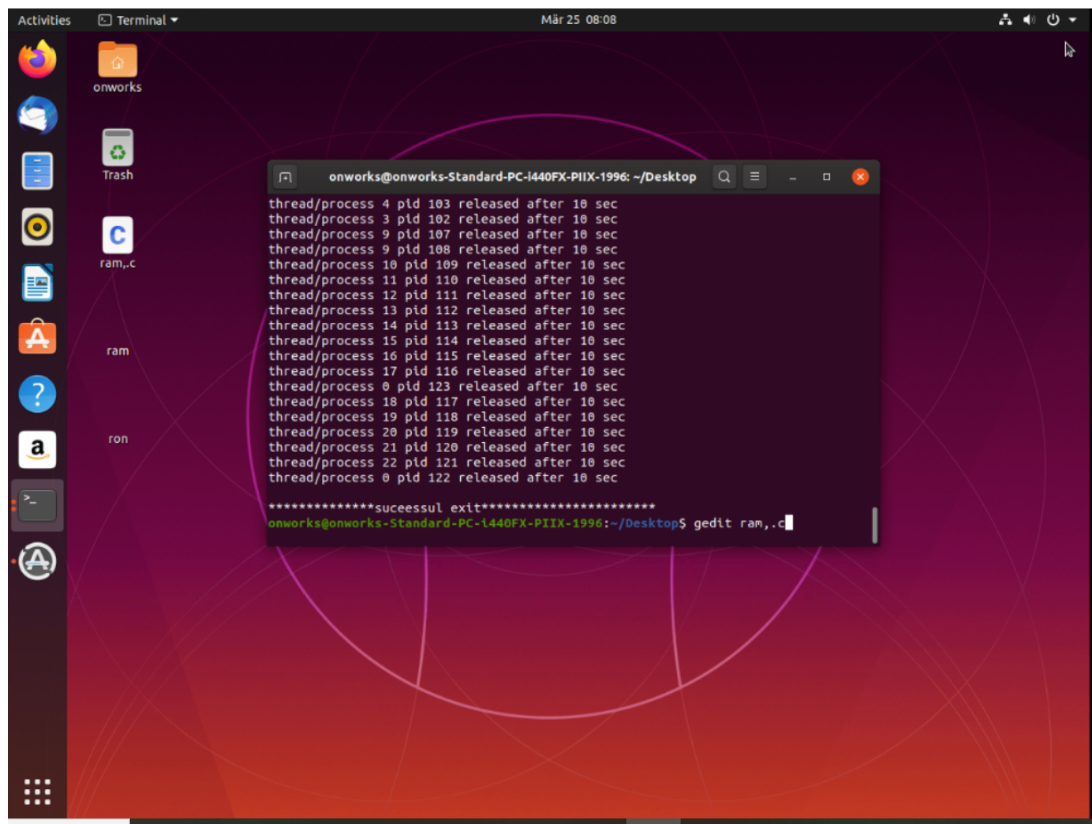
```
break;
```

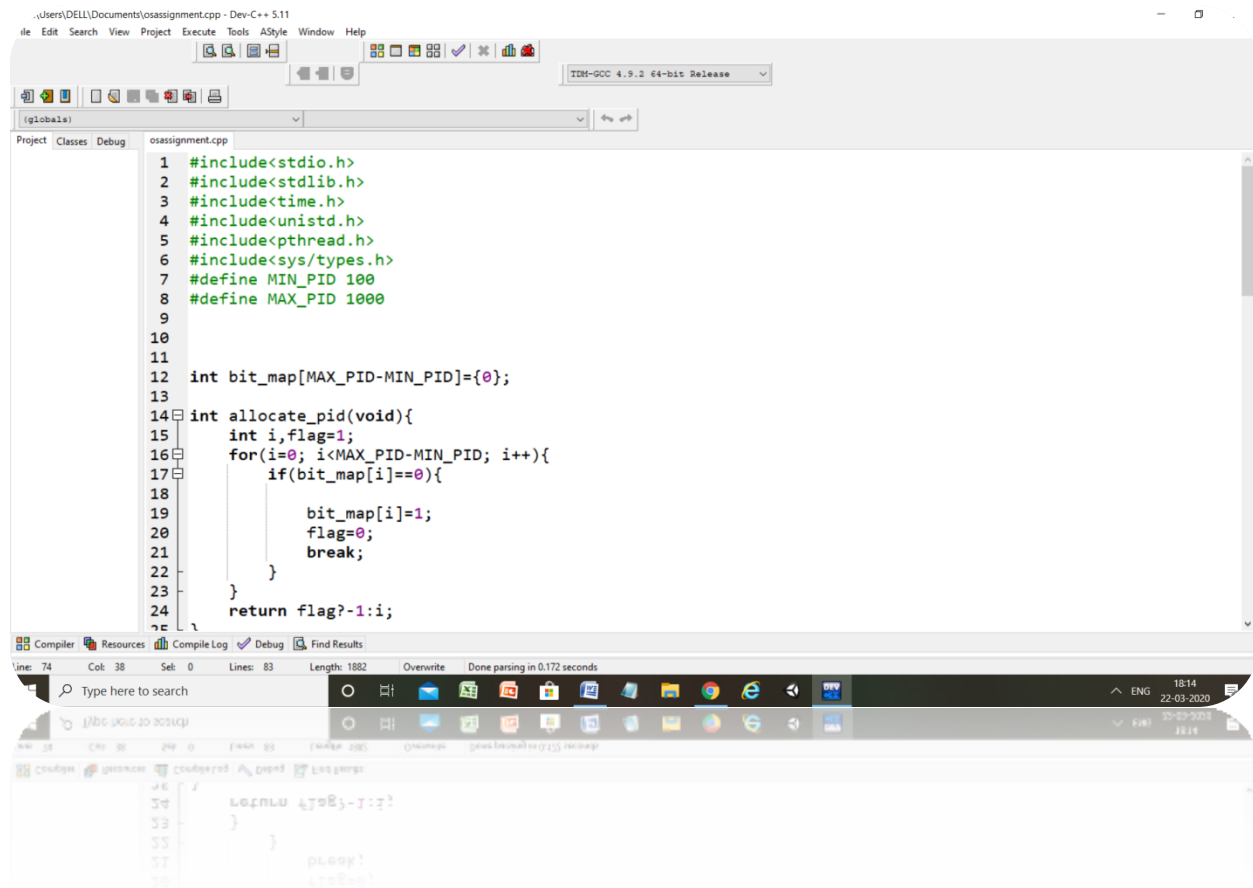
```
}
```

```
}
```



```
C:\Users\DELL\Documents\osassignment.cpp - Dev-C++ 5.11
File Edit Search View Project Execute Tools AStyle Window Help
TDM-GCC 4.9.2 64-bit Release
osassignment.cpp
22 }
23 }
24 return flag?-1:i;
25 }
26 void release_pid(int id){
27     bit_map[id]=0;
28 }
29 void *threading(void *az){
30     int tid = *((int*)az);
31
32     int id = allocate_pid();
33
34     if(id==-1){
35         printf("No PID available.");
36     }
37     else{
38         printf("Thread/Process %d is allocated with PID %d \n",tid,id+MIN_PID);
39
40         sleep(10);
41
42         printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
43     }
44 }
45
46 int main()
47 {
48     int NO_OF_Processes;
49     for(int i=0;i<NO_OF_Processes;i++)
50     {
51         int id;
52         id=fork();
53         printf("process with id is= %3d",id+MIN_PID);
54     }
55 }
56
57 int main()
58 {
59     int NO_OF_Processes;
60     for(int i=0;i<NO_OF_Processes;i++)
61     {
62         int id;
63         id=fork();
64         printf("process with id is= %3d",id+MIN_PID);
65     }
66 }
67
68 int main()
69 {
70     int NO_OF_Processes;
71     for(int i=0;i<NO_OF_Processes;i++)
72     {
73         int id;
74         id=fork();
75         printf("process with id is= %3d",id+MIN_PID);
76     }
77 }
78
79 int main()
80 {
81     int NO_OF_Processes;
82     for(int i=0;i<NO_OF_Processes;i++)
83     {
84         int id;
85         id=fork();
86         printf("process with id is= %3d",id+MIN_PID);
87     }
88 }
89
90 int main()
91 {
92     int NO_OF_Processes;
93     for(int i=0;i<NO_OF_Processes;i++)
94     {
95         int id;
96         id=fork();
97         printf("process with id is= %3d",id+MIN_PID);
98     }
99 }
100
101 int main()
102 {
103     int NO_OF_Processes;
104     for(int i=0;i<NO_OF_Processes;i++)
105     {
106         int id;
107         id=fork();
108         printf("process with id is= %3d",id+MIN_PID);
109     }
110 }
111
112 int main()
113 {
114     int NO_OF_Processes;
115     for(int i=0;i<NO_OF_Processes;i++)
116     {
117         int id;
118         id=fork();
119         printf("process with id is= %3d",id+MIN_PID);
120     }
121 }
122
123 int main()
124 {
125     int NO_OF_Processes;
126     for(int i=0;i<NO_OF_Processes;i++)
127     {
128         int id;
129         id=fork();
130         printf("process with id is= %3d",id+MIN_PID);
131     }
132 }
133
134 int main()
135 {
136     int NO_OF_Processes;
137     for(int i=0;i<NO_OF_Processes;i++)
138     {
139         int id;
140         id=fork();
141         printf("process with id is= %3d",id+MIN_PID);
142     }
143 }
144
145 int main()
146 {
147     int NO_OF_Processes;
148     for(int i=0;i<NO_OF_Processes;i++)
149     {
150         int id;
151         id=fork();
152         printf("process with id is= %3d",id+MIN_PID);
153     }
154 }
155
156 int main()
157 {
158     int NO_OF_Processes;
159     for(int i=0;i<NO_OF_Processes;i++)
160     {
161         int id;
162         id=fork();
163         printf("process with id is= %3d",id+MIN_PID);
164     }
165 }
166
167 int main()
168 {
169     int NO_OF_Processes;
170     for(int i=0;i<NO_OF_Processes;i++)
171     {
172         int id;
173         id=fork();
174         printf("process with id is= %3d",id+MIN_PID);
175     }
176 }
177
178 int main()
179 {
180     int NO_OF_Processes;
181     for(int i=0;i<NO_OF_Processes;i++)
182     {
183         int id;
184         id=fork();
185         printf("process with id is= %3d",id+MIN_PID);
186     }
187 }
188
189 int main()
190 {
191     int NO_OF_Processes;
192     for(int i=0;i<NO_OF_Processes;i++)
193     {
194         int id;
195         id=fork();
196         printf("process with id is= %3d",id+MIN_PID);
197     }
198 }
199
200 int main()
201 {
202     int NO_OF_Processes;
203     for(int i=0;i<NO_OF_Processes;i++)
204     {
205         int id;
206         id=fork();
207         printf("process with id is= %3d",id+MIN_PID);
208     }
209 }
210
211 int main()
212 {
213     int NO_OF_Processes;
214     for(int i=0;i<NO_OF_Processes;i++)
215     {
216         int id;
217         id=fork();
218         printf("process with id is= %3d",id+MIN_PID);
219     }
220 }
221
222 int main()
223 {
224     int NO_OF_Processes;
225     for(int i=0;i<NO_OF_Processes;i++)
226     {
227         int id;
228         id=fork();
229         printf("process with id is= %3d",id+MIN_PID);
230     }
231 }
232
233 int main()
234 {
235     int NO_OF_Processes;
236     for(int i=0;i<NO_OF_Processes;i++)
237     {
238         int id;
239         id=fork();
240         printf("process with id is= %3d",id+MIN_PID);
241     }
242 }
243
244 int main()
245 {
246     int NO_OF_Processes;
247     for(int i=0;i<NO_OF_Processes;i++)
248     {
249         int id;
250         id=fork();
251         printf("process with id is= %3d",id+MIN_PID);
252     }
253 }
254
255 int main()
256 {
257     int NO_OF_Processes;
258     for(int i=0;i<NO_OF_Processes;i++)
259     {
260         int id;
261         id=fork();
262         printf("process with id is= %3d",id+MIN_PID);
263     }
264 }
265
266 int main()
267 {
268     int NO_OF_Processes;
269     for(int i=0;i<NO_OF_Processes;i++)
270     {
271         int id;
272         id=fork();
273         printf("process with id is= %3d",id+MIN_PID);
274     }
275 }
276
277 int main()
278 {
279     int NO_OF_Processes;
280     for(int i=0;i<NO_OF_Processes;i++)
281     {
282         int id;
283         id=fork();
284         printf("process with id is= %3d",id+MIN_PID);
285     }
286 }
287
288 int main()
289 {
290     int NO_OF_Processes;
291     for(int i=0;i<NO_OF_Processes;i++)
292     {
293         int id;
294         id=fork();
295         printf("process with id is= %3d",id+MIN_PID);
296     }
297 }
298
299 int main()
300 {
301     int NO_OF_Processes;
302     for(int i=0;i<NO_OF_Processes;i++)
303     {
304         int id;
305         id=fork();
306         printf("process with id is= %3d",id+MIN_PID);
307     }
308 }
309
310 int main()
311 {
312     int NO_OF_Processes;
313     for(int i=0;i<NO_OF_Processes;i++)
314     {
315         int id;
316         id=fork();
317         printf("process with id is= %3d",id+MIN_PID);
318     }
319 }
320
321 int main()
322 {
323     int NO_OF_Processes;
324     for(int i=0;i<NO_OF_Processes;i++)
325     {
326         int id;
327         id=fork();
328         printf("process with id is= %3d",id+MIN_PID);
329     }
330 }
331
332 int main()
333 {
334     int NO_OF_Processes;
335     for(int i=0;i<NO_OF_Processes;i++)
336     {
337         int id;
338         id=fork();
339         printf("process with id is= %3d",id+MIN_PID);
340     }
341 }
342
343 int main()
344 {
345     int NO_OF_Processes;
346     for(int i=0;i<NO_OF_Processes;i++)
347     {
348         int id;
349         id=fork();
350         printf("process with id is= %3d",id+MIN_PID);
351     }
352 }
353
354 int main()
355 {
356     int NO_OF_Processes;
357     for(int i=0;i<NO_OF_Processes;i++)
358     {
359         int id;
360         id=fork();
361         printf("process with id is= %3d",id+MIN_PID);
362     }
363 }
364
365 int main()
366 {
367     int NO_OF_Processes;
368     for(int i=0;i<NO_OF_Processes;i++)
369     {
370         int id;
371         id=fork();
372         printf("process with id is= %3d",id+MIN_PID);
373     }
374 }
375
376 int main()
377 {
378     int NO_OF_Processes;
379     for(int i=0;i<NO_OF_Processes;i++)
380     {
381         int id;
382         id=fork();
383         printf("process with id is= %3d",id+MIN_PID);
384     }
385 }
386
387 int main()
388 {
389     int NO_OF_Processes;
390     for(int i=0;i<NO_OF_Processes;i++)
391     {
392         int id;
393         id=fork();
394         printf("process with id is= %3d",id+MIN_PID);
395     }
396 }
397
398 int main()
399 {
400     int NO_OF_Processes;
401     for(int i=0;i<NO_OF_Processes;i++)
402     {
403         int id;
404         id=fork();
405         printf("process with id is= %3d",id+MIN_PID);
406     }
407 }
408
409 int main()
410 {
411     int NO_OF_Processes;
412     for(int i=0;i<NO_OF_Processes;i++)
413     {
414         int id;
415         id=fork();
416         printf("process with id is= %3d",id+MIN_PID);
417     }
418 }
419
420 int main()
421 {
422     int NO_OF_Processes;
423     for(int i=0;i<NO_OF_Processes;i++)
424     {
425         int id;
426         id=fork();
427         printf("process with id is= %3d",id+MIN_PID);
428     }
429 }
430
431 int main()
432 {
433     int NO_OF_Processes;
434     for(int i=0;i<NO_OF_Processes;i++)
435     {
436         int id;
437         id=fork();
438         printf("process with id is= %3d",id+MIN_PID);
439     }
440 }
441
442 int main()
443 {
444     int NO_OF_Processes;
445     for(int i=0;i<NO_OF_Processes;i++)
446     {
447         int id;
448         id=fork();
449         printf("process with id is= %3d",id+MIN_PID);
450     }
451 }
452
453 int main()
454 {
455     int NO_OF_Processes;
456     for(int i=0;i<NO_OF_Processes;i++)
457     {
458         int id;
459         id=fork();
460         printf("process with id is= %3d",id+MIN_PID);
461     }
462 }
463
464 int main()
465 {
466     int NO_OF_Processes;
467     for(int i=0;i<NO_OF_Processes;i++)
468     {
469         int id;
470         id=fork();
471         printf("process with id is= %3d",id+MIN_PID);
472     }
473 }
474
475 int main()
476 {
477     int NO_OF_Processes;
478     for(int i=0;i<NO_OF_Processes;i++)
479     {
480         int id;
481         id=fork();
482         printf("process with id is= %3d",id+MIN_PID);
483     }
484 }
485
486 int main()
487 {
488     int NO_OF_Processes;
489     for(int i=0;i<NO_OF_Processes;i++)
490     {
491         int id;
492         id=fork();
493         printf("process with id is= %3d",id+MIN_PID);
494     }
495 }
496
497 int main()
498 {
499     int NO_OF_Processes;
500     for(int i=0;i<NO_OF_Processes;i++)
501     {
502         int id;
503         id=fork();
504         printf("process with id is= %3d",id+MIN_PID);
505     }
506 }
507
508 int main()
509 {
510     int NO_OF_Processes;
511     for(int i=0;i<NO_OF_Processes;i++)
512     {
513         int id;
514         id=fork();
515         printf("process with id is= %3d",id+MIN_PID);
516     }
517 }
518
519 int main()
520 {
521     int NO_OF_Processes;
522     for(int i=0;i<NO_OF_Processes;i++)
523     {
524         int id;
525         id=fork();
526         printf("process with id is= %3d",id+MIN_PID);
527     }
528 }
529
530 int main()
531 {
532     int NO_OF_Processes;
533     for(int i=0;i<NO_OF_Processes;i++)
534     {
535         int id;
536         id=fork();
537         printf("process with id is= %3d",id+MIN_PID);
538     }
539 }
540
541 int main()
542 {
543     int NO_OF_Processes;
544     for(int i=0;i<NO_OF_Processes;i++)
545     {
546         int id;
547         id=fork();
548         printf("process with id is= %3d",id+MIN_PID);
549     }
550 }
551
552 int main()
553 {
554     int NO_OF_Processes;
555     for(int i=0;i<NO_OF_Processes;i++)
556     {
557         int id;
558         id=fork();
559         printf("process with id is= %3d",id+MIN_PID);
560     }
561 }
562
563 int main()
564 {
565     int NO_OF_Processes;
566     for(int i=0;i<NO_OF_Processes;i++)
567     {
568         int id;
569         id=fork();
570         printf("process with id is= %3d",id+MIN_PID);
571     }
572 }
573
574 int main()
575 {
576     int NO_OF_Processes;
577     for(int i=0;i<NO_OF_Processes;i++)
578     {
579         int id;
580         id=fork();
581         printf("process with id is= %3d",id+MIN_PID);
582     }
583 }
584
585 int main()
586 {
587     int NO_OF_Processes;
588     for(int i=0;i<NO_OF_Processes;i++)
589     {
590         int id;
591         id=fork();
592         printf("process with id is= %3d",id+MIN_PID);
593     }
594 }
595
596 int main()
597 {
598     int NO_OF_Processes;
599     for(int i=0;i<NO_OF_Processes;i++)
600     {
601         int id;
602         id=fork();
603         printf("process with id is= %3d",id+MIN_PID);
604     }
605 }
606
607 int main()
608 {
609     int NO_OF_Processes;
610     for(int i=0;i<NO_OF_Processes;i++)
611     {
612         int id;
613         id=fork();
614         printf("process with id is= %3d",id+MIN_PID);
615     }
616 }
617
618 int main()
619 {
620     int NO_OF_Processes;
621     for(int i=0;i<NO_OF_Processes;i++)
622     {
623         int id;
624         id=fork();
625         printf("process with id is= %3d",id+MIN_PID);
626     }
627 }
628
629 int main()
630 {
631     int NO_OF_Processes;
632     for(int i=0;i<NO_OF_Processes;i++)
633     {
634         int id;
635         id=fork();
636         printf("process with id is= %3d",id+MIN_PID);
637     }
638 }
639
640 int main()
641 {
642     int NO_OF_Processes;
643     for(int i=0;i<NO_OF_Processes;i++)
644     {
645         int id;
646         id=fork();
647         printf("process with id is= %3d",id+MIN_PID);
648     }
649 }
650
651 int main()
652 {
653     int NO_OF_Processes;
654     for(int i=0;i<NO_OF_Processes;i++)
655     {
656         int id;
657         id=fork();
658         printf("process with id is= %3d",id+MIN_PID);
659     }
660 }
661
662 int main()
663 {
664     int NO_OF_Processes;
665     for(int i=0;i<NO_OF_Processes;i++)
666     {
667         int id;
668         id=fork();
669         printf("process with id is= %3d",id+MIN_PID);
670     }
671 }
672
673 int main()
674 {
675     int NO_OF_Processes;
676     for(int i=0;i<NO_OF_Processes;i++)
677     {
678         int id;
679         id=fork();
680         printf("process with id is= %3d",id+MIN_PID);
681     }
682 }
683
684 int main()
685 {
686     int NO_OF_Processes;
687     for(int i=0;i<NO_OF_Processes;i++)
688     {
689         int id;
690         id=fork();
691         printf("process with id is= %3d",id+MIN_PID);
692     }
693 }
694
695 int main()
696 {
697     int NO_OF_Processes;
698     for(int i=0;i<NO_OF_Processes;i++)
699     {
700         int id;
701         id=fork();
702         printf("process with id is= %3d",id+MIN_PID);
703     }
704 }
705
706 int main()
707 {
708     int NO_OF_Processes;
709     for(int i=0;i<NO_OF_Processes;i++)
710     {
711         int id;
712         id=fork();
713         printf("process with id is= %3d",id+MIN_PID);
714     }
715 }
716
717 int main()
718 {
719     int NO_OF_Processes;
720     for(int i=0;i<NO_OF_Processes;i++)
721     {
722         int id;
723         id=fork();
724         printf("process with id is= %3d",id+MIN_PID);
725     }
726 }
727
728 int main()
729 {
730     int NO_OF_Processes;
731     for(int i=0;i<NO_OF_Processes;i++)
732     {
733         int id;
734         id=fork();
735         printf("process with id is= %3d",id+MIN_PID);
736     }
737 }
738
739 int main()
740 {
741     int NO_OF_Processes;
742     for(int i=0;i<NO_OF_Processes;i++)
743     {
744         int id;
745         id=fork();
746         printf("process with id is= %3d",id+MIN_PID);
747     }
748 }
749
750 int main()
751 {
752     int NO_OF_Processes;
753     for(int i=0;i<NO_OF_Processes;i++)
754     {
755         int id;
756         id=fork();
757         printf("process with id is= %3d",id+MIN_PID);
758     }
759 }
760
761 int main()
762 {
763     int NO_OF_Processes;
764     for(int i=0;i<NO_OF_Processes;i++)
765     {
766         int id;
767         id=fork();
768         printf("process with id is= %3d",id+MIN_PID);
769     }
770 }
771
772 int main()
773 {
774     int NO_OF_Processes;
775     for(int i=0;i<NO_OF_Processes;i++)
776     {
777         int id;
778         id=fork();
779         printf("process with id is= %3d",id+MIN_PID);
780     }
781 }
782
783 int main()
784 {
785     int NO_OF_Processes;
786     for(int i=0;i<NO_OF_Processes;i++)
787     {
788         int id;
789         id=fork();
790         printf("process with id is= %3d",id+MIN_PID);
791     }
792 }
793
794 int main()
795 {
796     int NO_OF_Processes;
797     for(int i=0;i<NO_OF_Processes;i++)
798     {
799         int id;
800         id=fork();
801         printf("process with id is= %3d",id+MIN_PID);
802     }
803 }
804
805 int main()
806 {
807     int NO_OF_Processes;
808     for(int i=0;i<NO_OF_Processes;i++)
809     {
810         int id;
811         id=fork();
812         printf("process with id is= %3d",id+MIN_PID);
813     }
814 }
815
816 int main()
817 {
818     int NO_OF_Processes;
819     for(int i=0;i<NO_OF_Processes;i++)
820     {
821         int id;
822         id=fork();
823         printf("process with id is= %3d",id+MIN_PID);
824     }
825 }
826
827 int main()
828 {
829     int NO_OF_Processes;
830     for(int i=0;i<NO_OF_Processes;i++)
831     {
832         int id;
833         id=fork();
834         printf("process with id is= %3d",id+MIN_PID);
835     }
836 }
837
838 int main()
839 {
840     int NO_OF_Processes;
841     for(int i=0;i<NO_OF_Processes;i++)
842     {
843         int id;
844         id=fork();
845         printf("process with id is= %3d",id+MIN_PID);
846     }
847 }
848
849 int main()
850 {
851     int NO_OF_Processes;
852     for(int i=0;i<NO_OF_Processes;i++)
853     {
854         int id;
855         id=fork();
856         printf("process with id is= %3d",id+MIN_PID);
857     }
858 }
859
860 int main()
861 {
862     int NO_OF_Processes;
863     for(int i=0;i<NO_OF_Processes;i++)
864     {
865         int id;
866         id=fork();
867         printf("process with id is= %3d",id+MIN_PID);
868     }
869 }
870
871 int main()
872 {
873     int NO_OF_Processes;
874     for(int i=0;i<NO_OF_Processes;i++)
875     {
876         int id;
877         id=fork();
878         printf("process with id is= %3d",id+MIN_PID);
879     }
880 }
881
882 int main()
883 {
884     int NO_OF_Processes;
885     for(int i=0;i<NO_OF_Processes;i++)
886     {
887         int id;
888         id=fork();
889         printf("process with id is= %3d",id+MIN_PID);
890     }
891 }
892
893 int main()
894 {
895     int NO_OF_Processes;
896     for(int i=0;i<NO_OF_Processes;i++)
897     {
898         int id;
899         id=fork();
900         printf("process with id is= %3d",id+MIN_PID);
901     }
902 }
903
904 int main()
905 {
906     int NO_OF_Processes;
907     for(int i=0;i<NO_OF_Processes;i++)
908     {
909         int id;
910         id=fork();
911         printf("process with id is= %3d",id+MIN_PID);
912     }
913 }
914
915 int main()
916 {
917     int NO_OF_Processes;
918     for(int i=0;i<NO_OF_Processes;i++)
919     {
920         int id;
921         id=fork();
922         printf("process with id is= %3d",id+MIN_PID);
923     }
924 }
925
926 int main()
927 {
928     int NO_OF_Processes;
929     for(int i=0;i<NO_OF_Processes;i++)
930     {
931         int id;
932         id=fork();
933         printf("process with id is= %3d",id+MIN_PID);
934     }
935 }
936
937 int main()
938 {
939     int NO_OF_Processes;
940     for(int i=0;i<NO_OF_Processes;i++)
941     {
942         int id;
943         id=fork();
944         printf("process with id is= %3d",id+MIN_PID);
945     }
946 }
947
948 int main()
949 {
950     int NO_OF_Processes;
951     for(int i=0;i<NO_OF_Processes;i++)
952     {
953         int id;
954         id=fork();
955         printf("process with id is= %3d",id+MIN_PID);
956     }
957 }
958
959 int main()
960 {
961     int NO_OF_Processes;
962     for(int i=0;i<NO_OF_Processes;i++)
963     {
964         int id;
965         id=fork();
966         printf("process with id is= %3d",id+MIN_PID);
967     }
968 }
969
970 int main()
971 {
972     int NO_OF_Processes;
973     for(int i=0;i<NO_OF_Processes;i++)
974     {
975         int id;
976         id=fork();
977         printf("process with id is= %3d",id+MIN_PID);
978     }
979 }
980
981 int main()
982 {
983     int NO_OF_Processes;
984     for(int i=0;i<NO_OF_Processes;i++)
985     {
986         int id;
987         id=fork();
988         printf("process with id is= %3d",id+MIN_PID);
989     }
990 }
991
992 int main()
993 {
994     int NO_OF_Processes;
995     for(int i=0;i<NO_OF_Processes;i++)
996     {
997         int id;
998         id=fork();
999         printf("process with id is= %3d",id+MIN_PID);
1000     }
1001 }
1002
1003 int main()
1004 {
1005     int NO_OF_Processes;
1006     for(int i=0;i<NO_OF_Processes;i++)
1007     {
1008         int id;
1009         id=fork();
1010         printf("process with id is= %3d",id+MIN_PID);
1011     }
1012 }
1013
1014 int main()
1015 {
1016     int NO_OF_Processes;
1017     for(int i=0;i<NO_OF_Processes;i++)
1018     {
1019         int id;
1020         id=fork();
1021         printf("process with id is= %3d",id+MIN_PID);
1022     }
1023 }
1024
1025 int main()
1026 {
1027     int NO_OF_Processes;
1028     for(int i=0;i<NO_OF_Processes;i++)
1029     {
1030         int id;
1031         id=fork();
1032         printf("process with id is= %3d",id+MIN_PID);
1033     }
1034 }
1035
1036 int main()
1037 {
1038     int NO_OF_Processes;
1039     for(int i=0;i<NO_OF_Processes;i++)
1040     {
1041         int id;
1042         id=fork();
1043         printf("process with id is= %3d",id+MIN_PID);
1044     }
1045 }
1046
1047 int main()
1048 {
1049     int NO_OF_Processes;
1050     for(int i=0;i<NO_OF_Processes;i++)
1051     {
1052         int id;
1053         id=fork();
1054         printf("process with id is= %3d",id+MIN_PID);
1055     }
1056 }
1057
1058 int main()
1059 {
1060     int NO_OF_Processes;
1061     for(int i=0;i<NO_OF_Processes;i++)
1062     {
1063         int id;
1064         id=fork();
1065         printf("process with id is= %3d",id+MIN_PID);
1066     }
1067 }
1068
1069 int main()
1070 {
1071     int NO_OF_Processes;
1072     for(int i=0;i<NO_OF_Processes;i++)
1073     {
1074         int id;
1075         id=fork();
1076         printf("process with id is= %3d",id+MIN_PID);
1077     }
1078 }
1079
1080 int main()
1081 {
1082     int NO_OF_Processes;
1083     for(int i=0;i<NO_OF_Processes;i++)
1084     {
1085         int id;
1086         id=fork();
1087         printf("process with id is= %3d",id+MIN_PID);
1088     }
1089 }
1090
1091 int main()
1092 {
1093     int NO_OF_Processes;
1094     for(int i=0;i<NO_OF_Processes;i++)
1095     {
1096         int id;
1097         id=fork();
1098         printf("process with id is= %3d",id+MIN_PID);
1099     }
1100 }
1101
1102 int main()
1103 {
1104     int NO_OF_Processes;
1105     for(int i=0;i<NO_OF_Processes;i++)
1106     {
1107         int id;
1108         id=fork();
1109         printf("process with id is= %3d",id+MIN_PID);
1110     }
1111 }
1112
1113 int main()
1114 {
1115     int NO_OF_Processes;
1116     for(int i=0;i<NO_OF_Processes;i++)
1117     {
1118         int id;
1119         id=fork();
1120         printf("process with id is= %3d",id+MIN_PID);
1121     }
1122 }
1123
1124 int main()
1125 {
1126     int NO_OF_Processes;
1127     for(int i=0;i<NO_OF_Processes;i++)
1128     {
1129         int id;
1130         id=fork();
1131         printf("process with id is= %3d",id+MIN_PID);
1132     }
1133 }
1134
1135 int main()
1136 {
1137     int NO_OF_Processes;
1138     for(int i=0;i<NO_OF_Processes;i++)
1139     {
1140         int id;
1141         id=fork();
1142         printf("process with id is= %3d",id+MIN_PID);
1143     }
1144 }
1145
1146 int main()
1147 {
1148     int NO_OF_Processes;
1149     for(int i=0;i<NO_OF_Processes;i++)
1150     {
1151         int id;
1152         id=fork();
1153         printf("process with id is= %3d",id+MIN_PID);
1154     }
1155 }
1156
1157 int main()
1158 {
1159     int NO_OF_Processes;
1160     for(int i=0;i<NO_OF_Processes;i++)
1161     {
1162         int id;
1163         id=fork();
1164         printf("process with id is= %3d",id+MIN_PID);
1165     }
1166 }
1167
1168 int main()
1169 {
1170     int NO_OF_Processes;
1171     for(int i=0;i<NO_OF_Processes;i++)
1172     {
1173         int id;
1174         id=fork();
1175         printf("process with id is= %3d",id+MIN_PID);
1176     }
1177 }
1178
1179 int main()
1180 {
1181     int NO_OF_Processes;
1182     for(int i=0;i<NO_OF_Processes;i++)
1183     {
1184         int id;
1185         id=fork();
1186         printf("process with id is= %3d",id+MIN_PID);
1187     }
1188 }
1189
1190 int main()
1191 {
1192     int NO_OF_Processes;
1193     for(int i=0;i<NO_OF_Processes;i++)
1194     {
1195         int id;
1196         id=fork();
1197         printf("process with id is= %3d",id+MIN_PID);
1198     }
1199 }
1200
1201 int main()
1202 {
1203     int NO_OF_Processes;
1204     for(int i=0;i<NO_OF_Processes;i++)
1205     {
1206         int id;
1207         id=fork();
1208         printf("process with id is= %3d",id+MIN_PID);
1209     }
1210 }
1211
1212 int main()
1213 {
1214     int NO_OF_Processes;
1215     for(int i=0;i<NO_OF_Processes;i++)
1216     {
1217         int id;
1218         id=fork();
1219         printf("process with id is= %3d",id+MIN_PID);
1220     }
1221 }
1222
1223 int main()
1224 {
1225     int NO_OF_Processes;
1226     for(int i=0;i<NO_OF_Processes;i++)
1227     {
1228         int id;
1229         id=fork();
1230         printf("process with id is= %3d",id+MIN_PID);
1231     }
1232 }
1233
1234 int main()
1235 {
1236     int NO_OF_Processes;
1237     for(int i=0;i<NO_OF_Processes;i++)
1238     {
1239         int id;
1240         id=fork();
1241         printf("process with id is= %3d",id+MIN_PID);
1242     }
1243 }
1244
1245 int main()
1246 {
1247     int NO_OF_Processes;
1248     for(int i=0;i<NO_OF_Processes;i++)
1249     {
1250         int id;
1251         id=fork();
1252         printf("process with id is= %3d",id+MIN_PID);
1253     }
1254 }
1255
1256 int main()
1257 {
1258     int NO_OF_Processes;
1259     for(int i=0;i<NO_OF_Processes;i++)
1260     {
1261         int id;
1262         id=fork();
1263         printf("process with id is= %3d",id+MIN_PID);
1264     }
1265 }
1266
1267 int main()
1268 {
1269     int NO_OF_Processes;
1270     for(int i=0;i<NO_OF_Processes;i++)
1271     {
1272         int id;
1273         id=fork();
1274         printf("process with id is= %3d",id+MIN_PID);
1275     }
1276 }
1277
1278 int main()
1279 {
1280     int NO_OF_Processes;
1281     for(int i=0;i<NO_OF_Processes;i++)
1282     {
1283         int id;
1284         id=fork();
1285         printf("process with id is= %3d",id+MIN_PID);
1286     }
1287 }
1288
1289 int main()
1290 {
1291     int NO_OF_Processes;
1292     for(int i=0;i<NO_OF_Processes;i++)
1293     {
1294         int id;
1295         id=fork();
1296         printf("process with id is= %3d",id+MIN_PID);
1297     }
1298 }
1299
1300 int main()
1301 {
1302     int NO_OF_Processes;
1303     for(int i=0;i<NO_OF_Processes;i++)
1304     {
1305         int id;
1306         id=fork();
1307         printf("process with id is= %3d",id+MIN_PID);
1308     }
1309 }
1310
1311 int main()
1312 {
1313     int NO_OF_Processes;
1314     for(int i=0;i<NO_OF_Processes;i++)
1315     {
1316         int id;
1317         id=fork();
1318         printf("process with id is= %3d",id+MIN_PID);
1319     }
1320 }
1321
1322 int main()
1323 {
1324     int NO_OF_Processes;
1325     for(int i=0;i<NO_OF_Processes;i++)
1326     {
1327         int id;
1328         id=fork();
1329         printf("process with id is= %3d",id+MIN_PID);
1330     }
1331 }
1332
1333 int main()
```



The screenshot shows the Dev-C++ IDE with a C++ program named 'osassignment.cpp'. The code includes headers for stdio, stdlib, time, unistd, pthread, and sys/types. It defines MIN_PID as 100 and MAX_PID as 1000. A global array 'bit_map' of size MAX_PID - MIN_PID is initialized to zero. The 'allocate_pid' function iterates from MIN_PID to MAX_PID, finding the first available PID (where bit_map[i] is 0), sets it to 1, and returns the PID. The status bar at the bottom indicates 'Line: 74, Col: 38, Sel: 0, Lines: 83, Length: 1882, Overwrite, Done parsing in 0.172 seconds'.

```
1 #include<stdio.h>
2 #include<stdlib.h>
3 #include<time.h>
4 #include<unistd.h>
5 #include<pthread.h>
6 #include<sys/types.h>
7 #define MIN_PID 100
8 #define MAX_PID 1000
9
10
11
12 int bit_map[MAX_PID-MIN_PID]={0};
13
14 int allocate_pid(void){
15     int i,flag=1;
16     for(i=0; i<MAX_PID-MIN_PID; i++){
17         if(bit_map[i]==0){
18
19             bit_map[i]=1;
20             flag=0;
21             break;
22         }
23     }
24     return flag?-1:i;
25 }
```

DESCRIPTION:-

Use of each header file:

Include<stdio.h> :- This stdio.h header defines various variable types, macros as well as various functions for performing Input output operations.

Include<stdlib.h> :- This is library file for standard library

Include<pthread.h> :- header file used to use all the fuctions of pthread library

Include<time.h> :- the time.h header file contains definitions of function to get and manipulate date and time information.

Include <sys/types.h> :- defines various data types which are used in system source code.

In this project firstly two constants are declared which are used to assign the range of pid such that process identifier must lie between the given range .

define MIN_PID used to assign minimum range from which the pid is assigned to process or threads

define MAX_PID used to assign maximum range upto which pid is assigned.

Bit_map : preallocated bitmap function is used

Int bit_map[MAX_PID-MIN_PID]={0}

If(bit_map[i]==0) :- means pid available

Else : pid is in use

(If bit_map[i]==0) :- pid available and next line will be in critical section with a mutex lock whichever thread gets the mutex lock get the pid.

Int allocate_map(void) : this fuction is created to initialize bitmap function

Int allocate_pid(void): allocates and return the pid . it will check the condition if

It will return -1 it means all pid are in use and none is free to allocate and if it return 1 it means pid is available and next line should be in critical section with a mutex lock whichever thread gets the lock gets the pid // crtical section with mutex//

Void release_pid(int id) :- releases the pid means releases the mutex lock and come out of the critical section so that the released PID is assign to new process.

Process can be created with the help of fork() system call but here it is specify that int allocate_pid(void) is used to provide id to all the processes.

Void *threading(void *az):- is used to cerate multithreaded system which assign PID to each thread int id=allocate_pid() will allocate unique pid to each and every process such that no process has same id. After assigning pid to every thread . each thread sleep for random amount of time using sleep function. And after that pid is released.

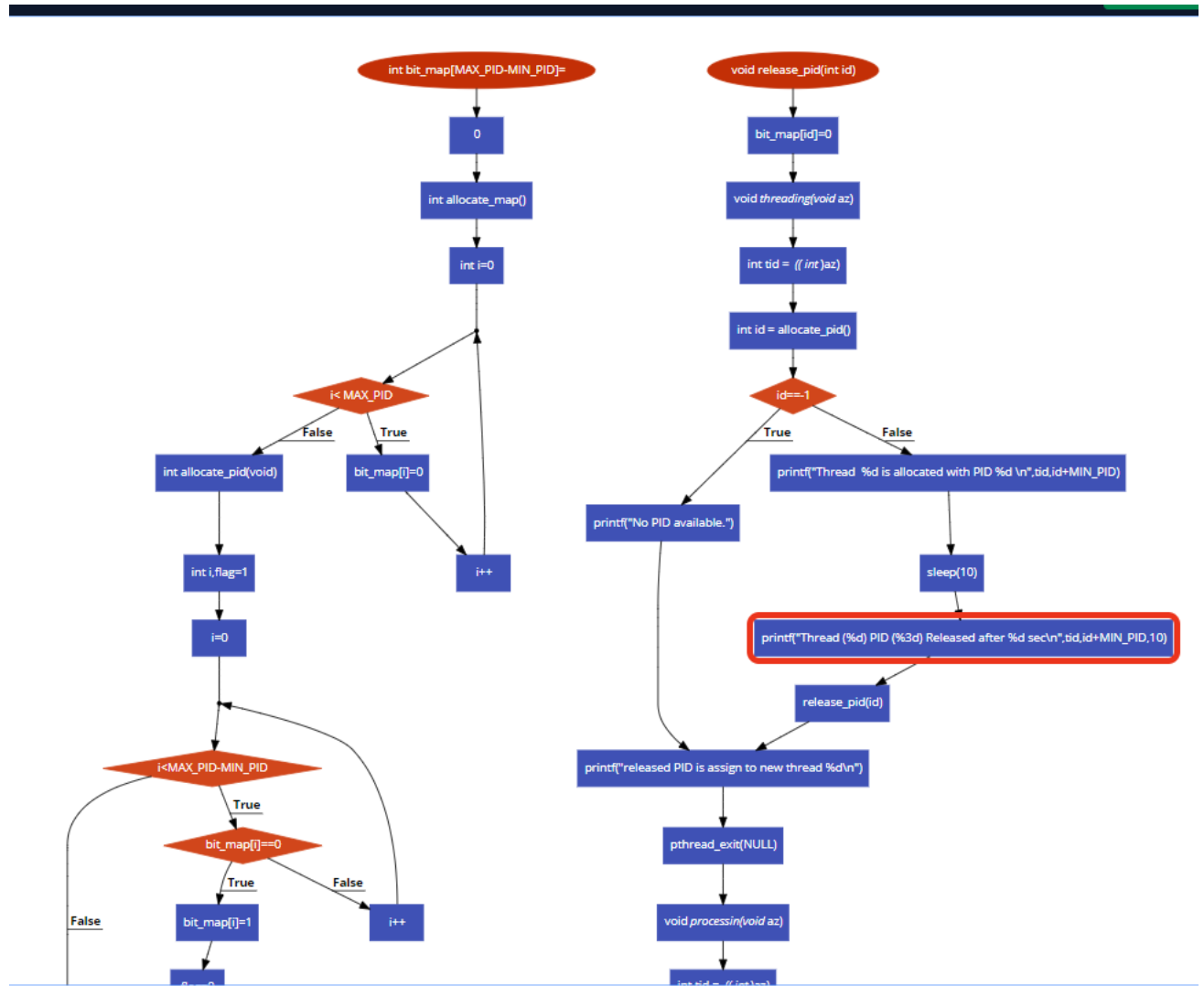
pthread_create ():- it is a function used to create a new thread with attributed specified by attribute within a processs . if attribute is null, the default attribute are used by pthread_create function()

pthread_join():- This function is basically use to provide a simple mechanism which allow an application to wait for a thread to terminate.

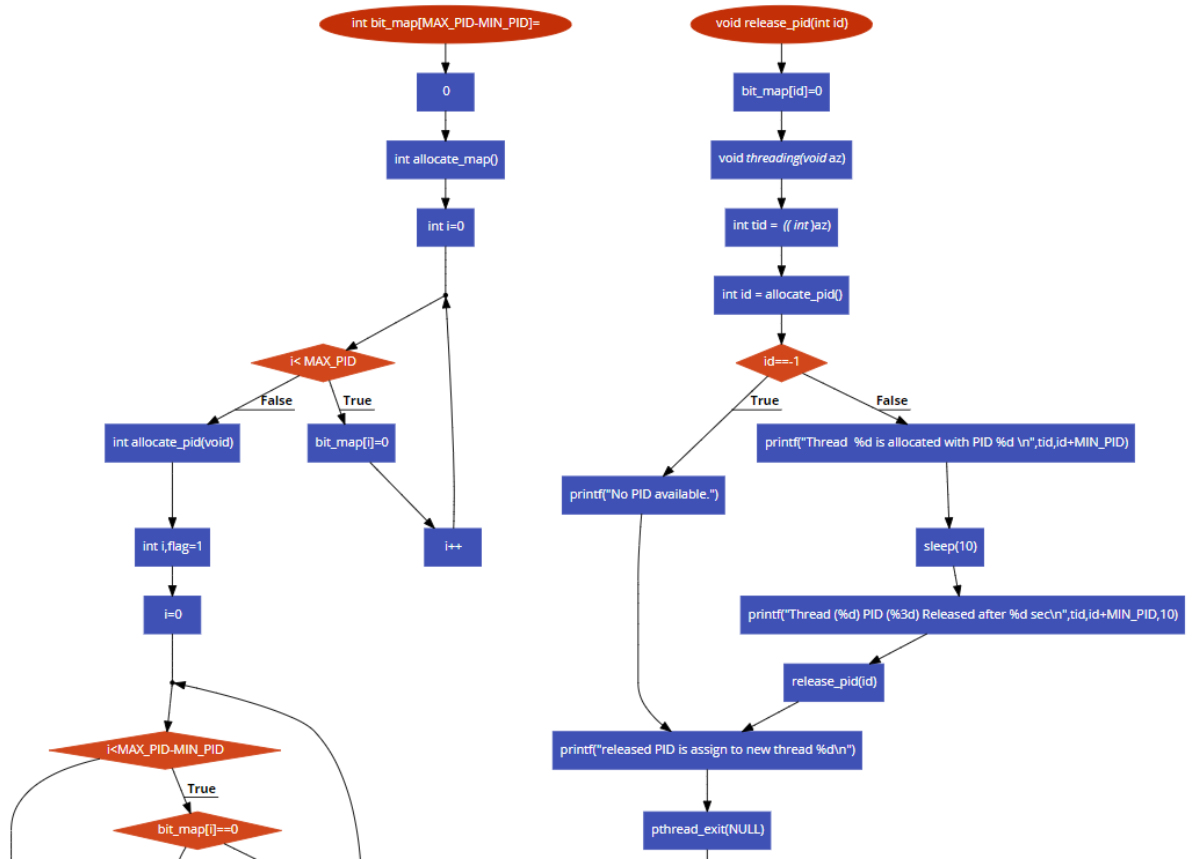
ALGORITHM USED:-FCFS (FIRST COME FIRST SERVE) :-It is the simplest form of process scheduling algorithm in which I/O requests are served according to the process arrival. The request arrives first will be accessed and served first .Since it follows the order of arrival,the process first allocate a unique pid and then other process according to their order of arrival. And process will release id also on the basis of their arrival.In his type of algorithm processes that requests CPU first get the allocation first.This is managed with the help of FIFO queue. For eg if we create Five processes p1,p2,p3,p4,p5 and so on each process is assign with PID according to their arrival in FIFO queue and released PID the same manner same is applicabe for thread also.

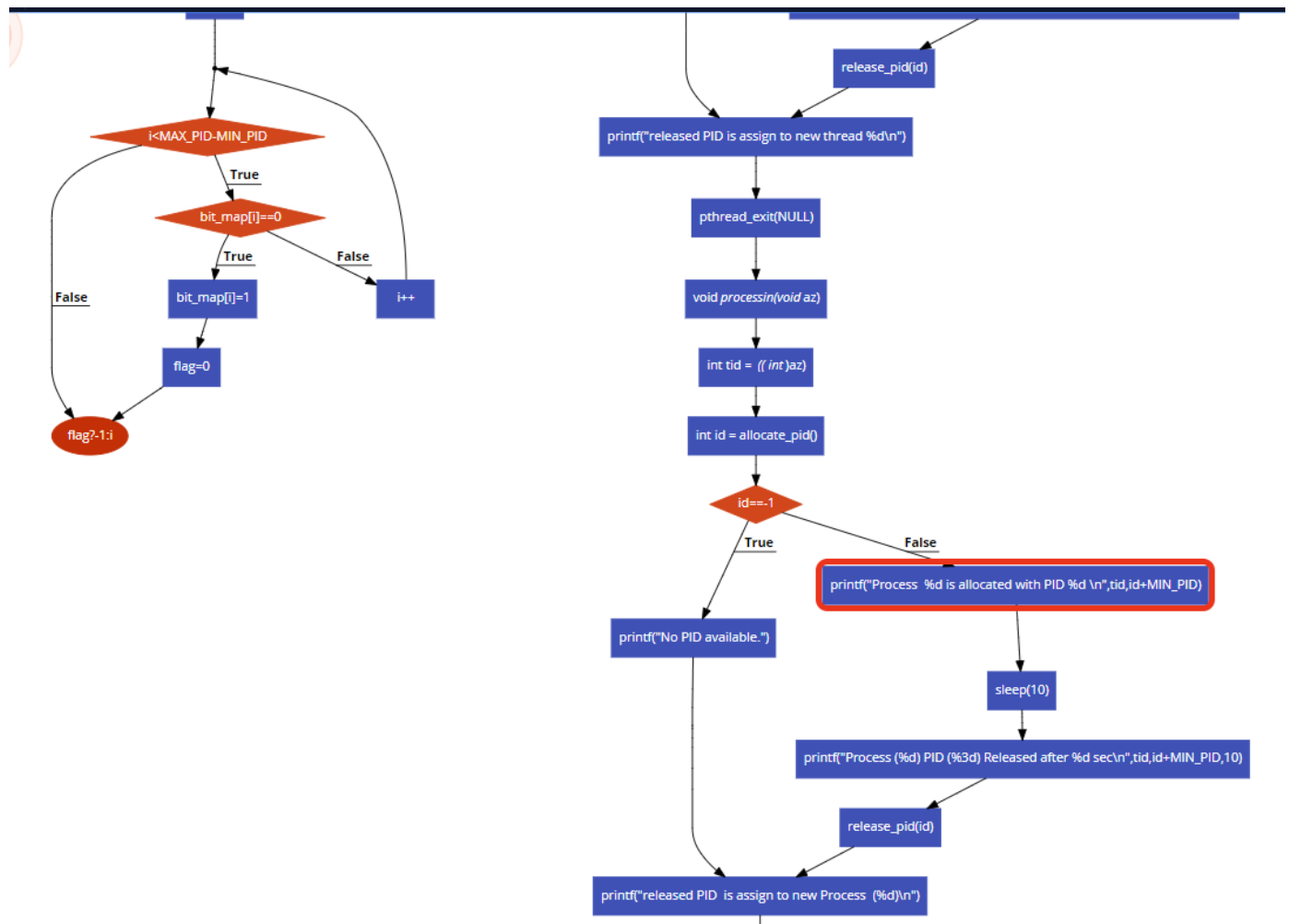
LOCKS :- A Lock is basically used to limit the access of resource when many threads wants to access that resources . it basically based on mutual exclusion which ever thread gets the mutex lock that thread enter into mutual exclusion and that time no other thread can enter into the critical section.

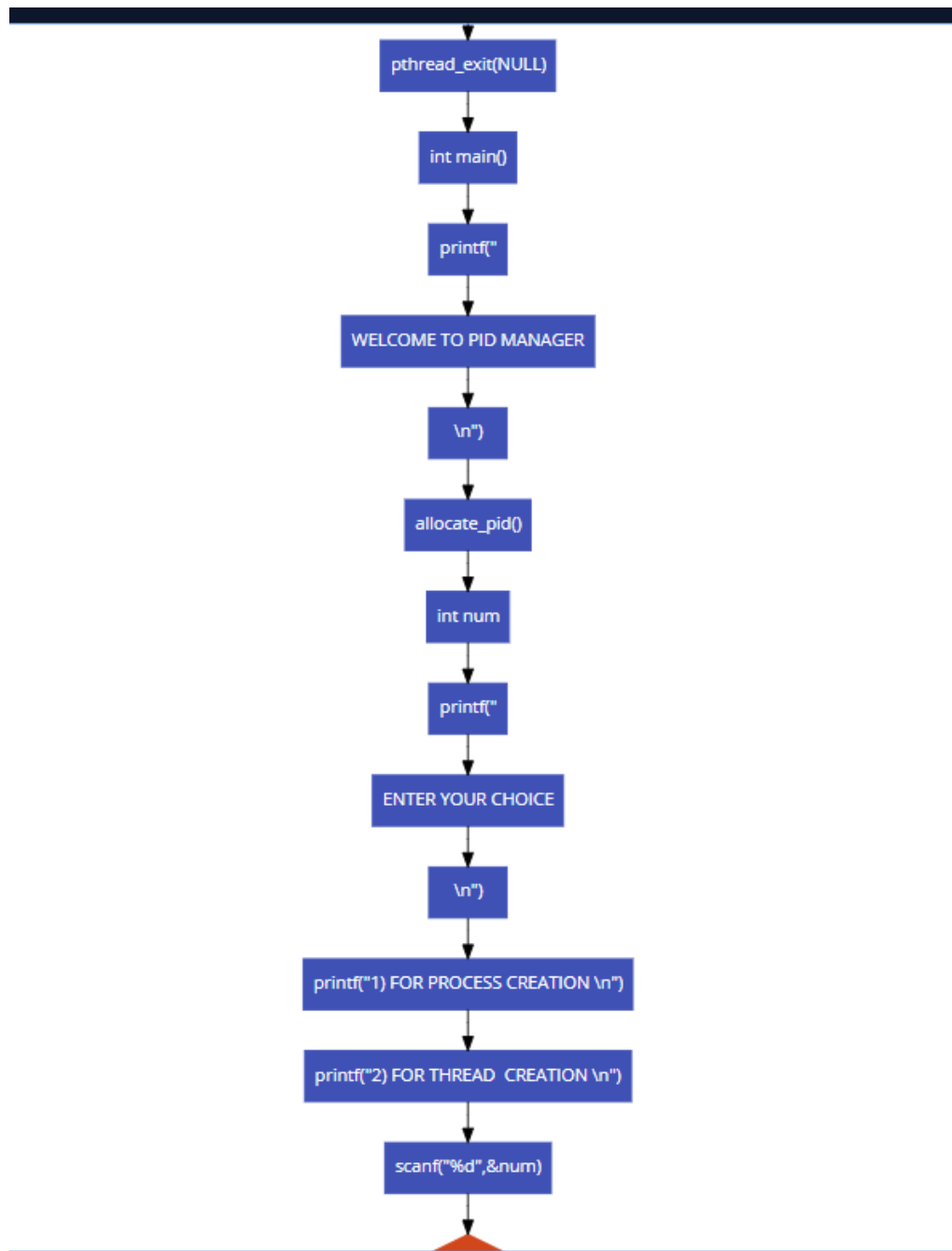
ALGORITHM – IN FROM OF FLOW CHART

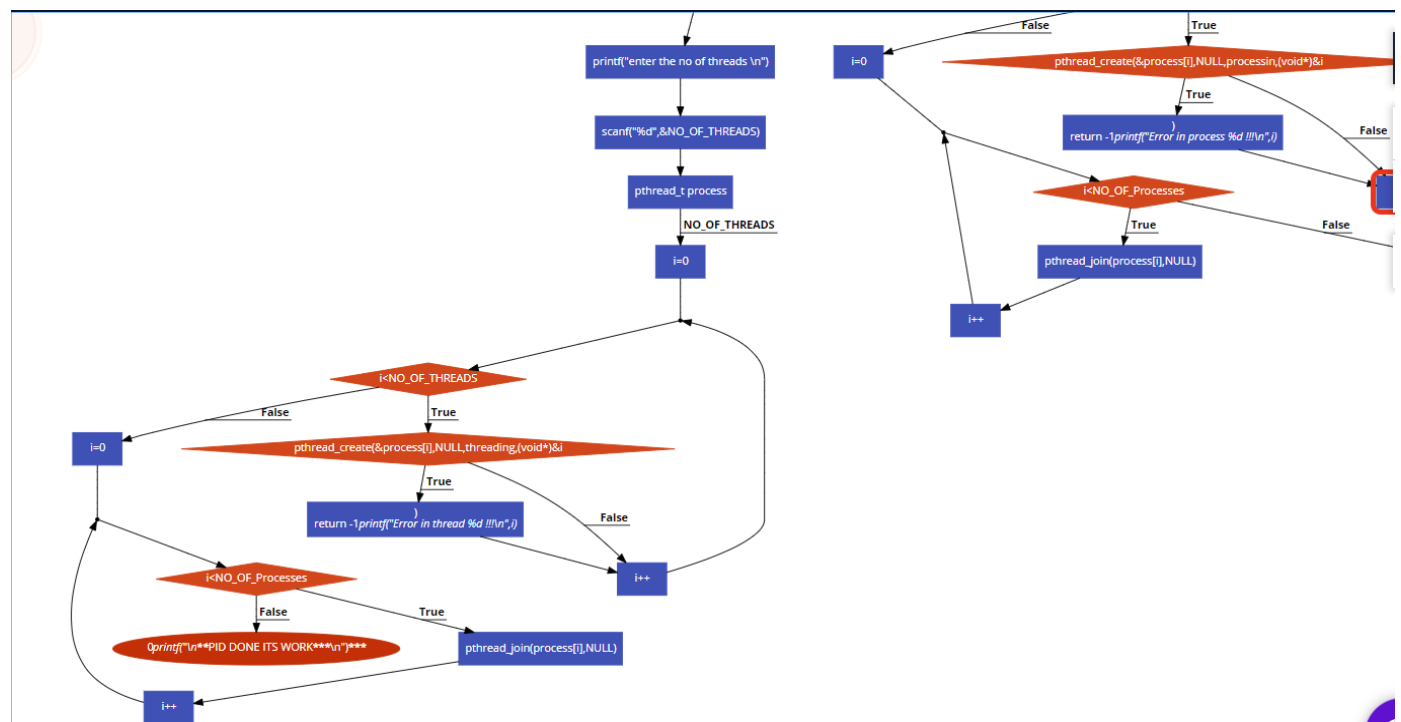
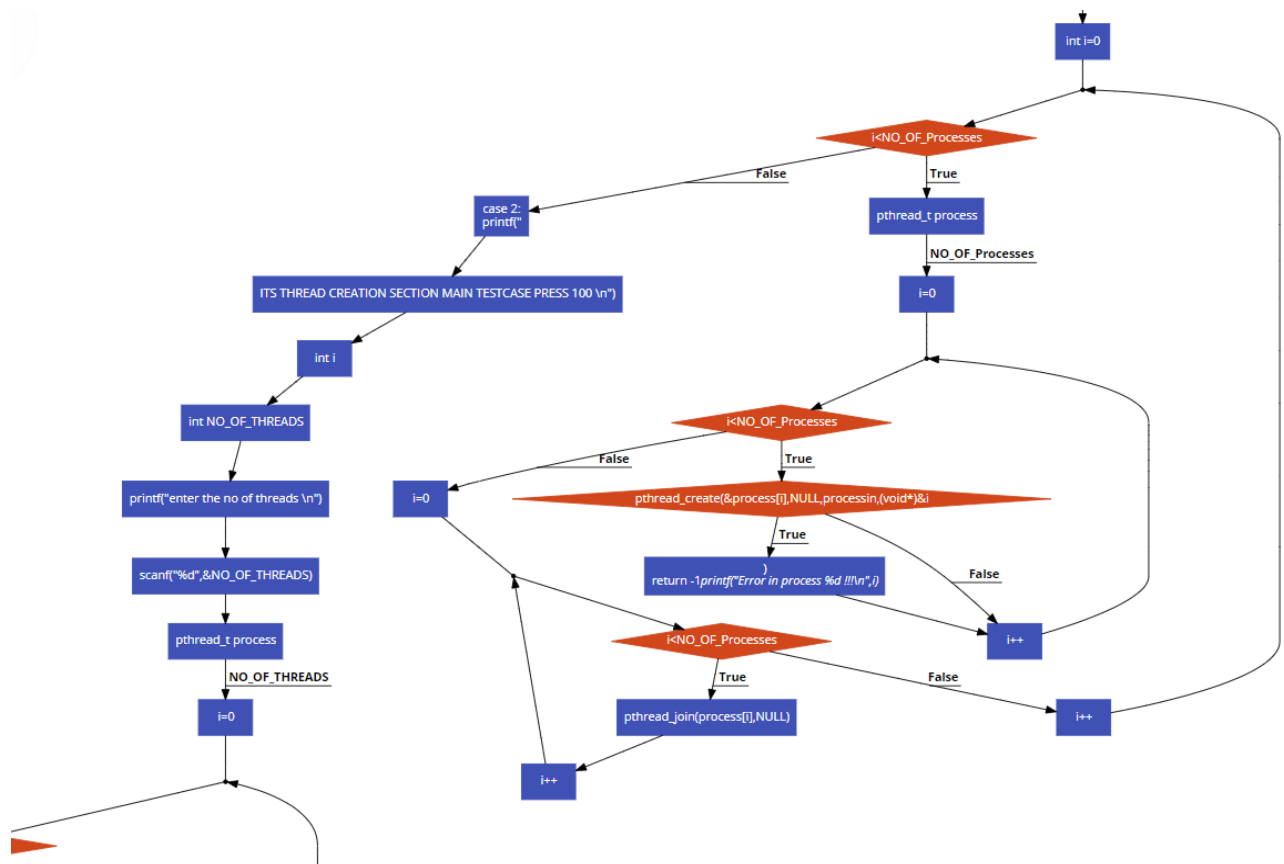












COMPLEXITY TABLE :-

CYCLOMATIC COMPLEXITY OF EACH FUNCTION USED IN CODE :-

FUNCTION NAME	CYCLOMATIC COMPLEXITY	NLOC
allocate_map()	2	5
allocate_pid()	4	11
release_pid()	1	3
Threading	2	15
Main	6	31

TIME COMPLEXITY :- allocate_map(),allocate_pid(),release_pid() has complexity constant because it will depend upon NO_OF_PROCESSES WHICH ARE CONSTANT whereas main() function has $O(n)$ complexity . **therefore overall complexity is $O(n)$.**

FORMULA USED IN CALCULATION:- $E-N+2P$

E= NO OF EDGES

N= NO OF NODES

P= NO OF CONNECTED COMPONENTS IN GRAPH

CONSTRAINT GIVEN IN PROBLEM :- use the following function for obtaining and releasing a pid :

- int allocate_map(void)—Creates and used for initialization of a data structure for representing pids;

Returns —1 if unsuccessful, 1 if successful

- int allocate_pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate a pid (all pids are in use)

- void release_pid(int pid)—Releases a pid

Modify the above problem by creating a multithreaded program that examine your solution.

You will create a number of threads—for example, 100—and each thread will allocate with a pid, which is unique

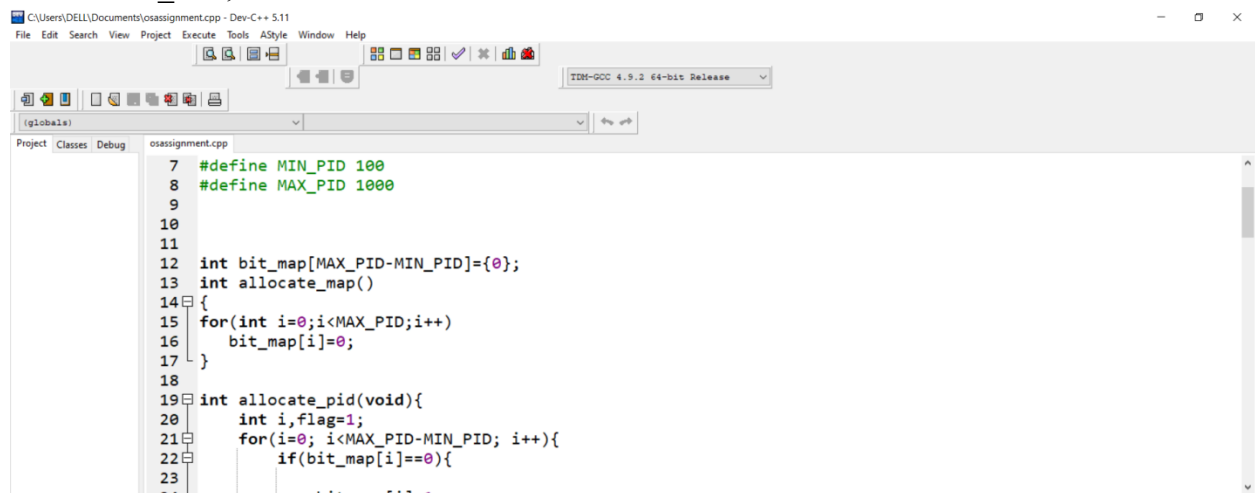
sleep for a random period of time, and then release the pid .The release pid is assign to new process.

SOLUTION TO CONSTRAINT :-

- int allocate_pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate a pid (all pids are in use)

CONSTRAINT :- • int allocate_map(void)—Creates and initializes a data structure for representing pids;

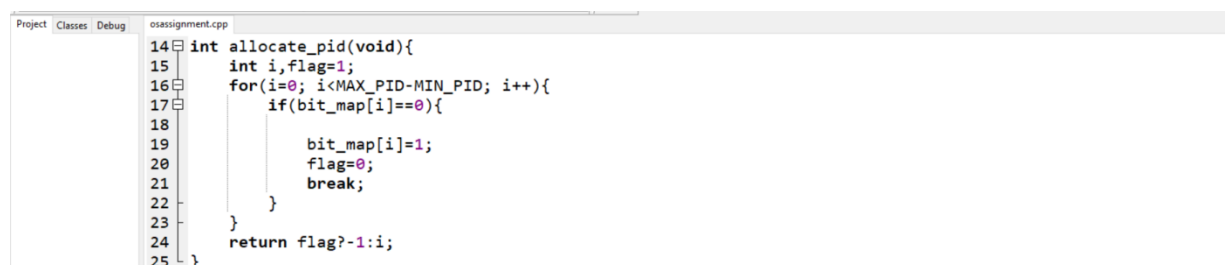
Solution :- bitmap() is used to initialize and create pid's whose range lie between MIN_PID AND MAX_PID;



```
7 #define MIN_PID 100
8 #define MAX_PID 1000
9
10
11
12 int bit_map[MAX_PID-MIN_PID]={0};
13 int allocate_map()
14 {
15     for(int i=0;i<MAX_PID;i++)
16         bit_map[i]=0;
17 }
18
19 int allocate_pid(void){
20     int i,flag=1;
21     for(i=0; i<MAX_PID-MIN_PID; i++){
22         if(bit_map[i]==0){
23             bit_map[i]=1;
24             flag=0;
25             break;
26         }
27     }
28     return flag?-1:i;
```

- int allocate_pid(void)—Allocates and returns a pid; returns— 1 if unable to allocate a pid (all pids are in use)

CODE SNIPPET :-



```
14 int allocate_pid(void){
15     int i,flag=1;
16     for(i=0; i<MAX_PID-MIN_PID; i++){
17         if(bit_map[i]==0){
18             bit_map[i]=1;
19             flag=0;
20             break;
21         }
22     }
23     return flag?-1:i;
24 }
25 }
```

- You will create a number of threads—for example, 100—and each thread will request a pid, and sleep for random amount of time.

```

29 void *threading(void *az){
30     int tid = *((int*)az);
31
32
33     int id = allocate_pid();
34
35
36     if(id==-1){
37         printf("No PID available.");
38     }
39     else{
40         printf("Thread/Process  %d is allocated with PID %d \n",tid,id+MIN_PID);
41
42
43
44         sleep(10);
45
46         printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
47         release_pid(id);
48
49     }
50     printf("released PID  is assign to new thread/Process  (%d)\n");
51     pthread_exit(NULL);
52
53 }

```

```
60 }  
61 }****/  
62 int main(){  
63     printf(">>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>\n");  
64     allocate_pid();  
65     // process_creation()  
66  
67     int i;  
68     int NO_OF_Processes;  
69     printf("enter the no of processes/threads \n");  
70     scanf("%d",&NO_OF_Processes);  
71  
72     pthread_t process[NO_OF_Processes];  
73     srand(time(NULL));  
74     for(i=0; i<NO_OF_Processes; i++){  
75         if(pthread_create(&process[i],NULL,threading,(void*)&i))  
76             return -1*printf("Error in thread %d !!!\n",i);  
77     }  
78 }
```

```
#define MAX_PID 1000
```

```

7  #define MIN_PID 100
8  #define MAX_PID 1000
9
10
11
12  int bit_map[MAX_PID-MIN_PID]={0};
13

```

BOUNDARY CONDITIONS :- BOUNDARY CONDITION:-PID RANGE SHOULD LIE BETWEEN 100-1000 and each process must have different PID;

This code snippet ensure that PID must lie between 100-1000 and each process have different PID

We have used bit_map[] character array which is used to assign pids .This character array ensure that pid must lie between MAX_PID AND MIN_PID

MAX_PID IS UPTO 1000

MIN_PID IS INITIALIZE 100

SO bit_map[] have range between 100-1000

```

7  #define MIN_PID 100
8  #define MAX_PID 1000
9
10
11
12  int bit_map[MAX_PID-MIN_PID]={0};
13

```

```

45
46
47  printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
48  release_pid(id);

```

SECOND BOUNDARY CONDITION :- EACH PROCESS/THREAD SHOULD ASSIGNED WITH UNIQUE PID.

We have create a function as

Void *threading(void *az)

{ int id=*((int *)az); // his will initialize id for hreads

Int id= allocate_pid() // it will call he allocate_pid() method o allocate ids

Printf(hread/process %d is allocated with pid %d,tid,id+MIN_PID)

// this printf function allocate unique pid which are taken from allocate_pid() and added o min_pid,

```
29 void *threading(void *az){
30     int tid = *((int*)az);
31
32
33     int id = allocate_pid();
34
35
36     if(id==-1){
37         printf("No PID available.");
38     }
39     else{
40         printf("Thread/Process %d is allocated with PID %d \n",tid,id+MIN_PID);
41
42
43
44         sleep(10);
45
46         printf("Thread/Process (%d) PID (%3d) Released after %d sec\n",tid,id+MIN_PID,10);
47         release_pid(id);
48     }
49
50     printf("released PID is assign to new thread/Process (%d)\n");
51     pthread_exit(NULL);
52 }
```

[illegible]

Additional idea used :- I have create a switch case which asked the user to create process or thread if user click 1 he enter into process creation section and if he enter 2 he thread

creation section.

[illegible]

He enter 1 so he will enter in process creation section..and if he press 2 he will enter in hread creation section

[illegible]

----- : TESTCASES:-----

TESTCASE 1 : Output of PID must lie between 100-1000 PID

[illegible]

TEST CASE :- PID SHOULD BE UNIQUE EACH PID SHOULD HAVE DIFFERENT VALUE :- for eg :- here are 5 processess :-

4 with PID 101

4 with PID 102

4 with PID 103

4 with PID 104

0 with PID 105

[illegible]

TESTCASE 3 :- Create 100 threads allocate unique pid to each and then sleep for random amount of time and release its pid. Released pid is assigned to new process/thread.

```
>>>>>>>>>>>>>>>>>>>WELCOME TO PID MANAGER>>>>>>>>>>>>>>
```

enter the no of processes/threads

5

```
Thread/Process 3 is allocated with PID 101
```

```
Thread/Process 3 is allocated with PID 102
```

```
Thread/Process 3 is allocated with PID 103
```

```
Thread/Process 0 is allocated with PID 104
```

```
Thread/Process 0 is allocated with PID 105
```

```
Thread/Process (3) PID (101) Released after 10 sec
```

```
released PID is assign to new thread/Process (4)
```

```
Thread/Process (0) PID (105) Released after 10 sec
```

```
released PID is assign to new thread/Process (20)
```

```
Thread/Process (3) PID (103) Released after 10 sec
```

```
released PID is assign to new thread/Process (12)
```

```
Thread/Process (0) PID (104) Released after 10 sec
```

```
released PID is assign to new thread/Process (16)
```

```
Thread/Process (3) PID (102) Released after 10 sec
```

```
released PID is assign to new thread/Process (8)
```

*****SUCCESSFUL EXIT*****

```
Process exited after 26.39 seconds with return value 0
```

```
Press any key to continue . . .
```

TESTCASE :-

[illegible]

• • • • •

100 is given as input

[illegible]

**OUTPUT :- IT WILL DISPLAY 100 HREADS WITH HEIR
CORRESPONDING PIDS WHICH ARE UNIQUE..**

```
C:\Users\DELL\Documents\osassignment.exe
Thread/Process 52 is allocated with PID 152
Thread/Process 54 is allocated with PID 153
Thread/Process 55 is allocated with PID 154
Thread/Process 18 is allocated with PID 118
Thread/Process 0 is allocated with PID 197
Thread/Process 58 is allocated with PID 157
Thread/Process 59 is allocated with PID 158
Thread/Process 61 is allocated with PID 159
Thread/Process 62 is allocated with PID 160
Thread/Process 62 is allocated with PID 161
Thread/Process 62 is allocated with PID 162
Thread/Process 64 is allocated with PID 163
Thread/Process 38 is allocated with PID 137
Thread/Process 66 is allocated with PID 165
Thread/Process 38 is allocated with PID 138
Thread/Process 68 is allocated with PID 167
Thread/Process 69 is allocated with PID 168
Thread/Process 71 is allocated with PID 169
Thread/Process 72 is allocated with PID 170
Thread/Process 74 is allocated with PID 171
Thread/Process 75 is allocated with PID 172
Thread/Process 75 is allocated with PID 173
Thread/Process 77 is allocated with PID 174
Thread/Process 78 is allocated with PID 175
Thread/Process 78 is allocated with PID 176
Thread/Process 80 is allocated with PID 177
Thread/Process 47 is allocated with PID 146
Thread/Process 80 is allocated with PID 179
Thread/Process 82 is allocated with PID 180
Thread/Process 83 is allocated with PID 181
Thread/Process 0 is allocated with PID 182
Thread/Process 0 is allocated with PID 183
Thread/Process 0 is allocated with PID 184
Thread/Process 0 is allocated with PID 185
Thread/Process 0 is allocated with PID 186
Thread/Process 0 is allocated with PID 187
Thread/Process 0 is allocated with PID 188
Thread/Process 0 is allocated with PID 189
Thread/Process 0 is allocated with PID 190
Thread/Process 0 is allocated with PID 191
Thread/Process 0 is allocated with PID 192
Thread/Process 0 is allocated with PID 193
Thread/Process 0 is allocated with PID 194
Thread/Process 0 is allocated with PID 195
Thread/Process 0 is allocated with PID 196
Thread/Process 58 is allocated with PID 156
Thread/Process 0 is allocated with PID 198
Thread/Process 0 is allocated with PID 199
Thread/Process 0 is allocated with PID 200
```

**THREADS SLEEP FOR 10 SEC HAN PID ARE RELEASED BY HRADS
AFTER 10 SEC**

RELEASED PID ARE ASSIGNED TO NEW PROCESS.

```
Thread/Process (66) PID (165) Released after 10 sec
released PID is assign to new thread/Process (260)
Thread/Process (38) PID (138) Released after 10 sec
released PID is assign to new thread/Process (152)
released PID is assign to new thread/Process (64)
Thread/Process (69) PID (168) Released after 10 sec
released PID is assign to new thread/Process (272)
Thread/Process (71) PID (169) Released after 10 sec
released PID is assign to new thread/Process (276)
Thread/Process (15) PID (114) Released after 10 sec
released PID is assign to new thread/Process (56)
Thread/Process (75) PID (172) Released after 10 sec
released PID is assign to new thread/Process (288)
Thread/Process (77) PID (174) Released after 10 sec
released PID is assign to new thread/Process (296)
Thread/Process (78) PID (176) Released after 10 sec
released PID is assign to new thread/Process (304)
Thread/Process (47) PID (146) Released after 10 sec
released PID is assign to new thread/Process (184)
Thread/Process (82) PID (180) Released after 10 sec
released PID is assign to new thread/Process (320)
Thread/Process (83) PID (181) Released after 10 sec
released PID is assign to new thread/Process (324)
Thread/Process (0) PID (182) Released after 10 sec
released PID is assign to new thread/Process (328)
Thread/Process (0) PID (184) Released after 10 sec
released PID is assign to new thread/Process (336)
Thread/Process (0) PID (185) Released after 10 sec
released PID is assign to new thread/Process (340)
Thread/Process (21) PID (120) Released after 10 sec
released PID is assign to new thread/Process (80)
Thread/Process (0) PID (186) Released after 10 sec
released PID is assign to new thread/Process (344)
Thread/Process (0) PID (188) Released after 10 sec
released PID is assign to new thread/Process (352)
Thread/Process (0) PID (190) Released after 10 sec
released PID is assign to new thread/Process (360)
Thread/Process (0) PID (191) Released after 10 sec
released PID is assign to new thread/Process (364)
Thread/Process (29) PID (128) Released after 10 sec
released PID is assign to new thread/Process (112)
Thread/Process (0) PID (193) Released after 10 sec
released PID is assign to new thread/Process (372)
Thread/Process (0) PID (194) Released after 10 sec
released PID is assign to new thread/Process (376)
Thread/Process (0) PID (195) Released after 10 sec
released PID is assign to new thread/Process (380)
Thread/Process (58) PID (156) Released after 10 sec
released PID is assign to new thread/Process (224)
```

NO OF COMMITS OR REVISIONS IN CODE :-

ronit-lab / os-316assignment

Watch 0

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

Branch: master

Commits on Mar 23, 2020

Merge pull request #2 from ronit-lab/ronit-lab-patch-2

ronit-lab committed 41 seconds ago

Verified

74ba6f3

<>

add allocate_map() which will initialize the bit_map character array

ronit-lab committed 1 minute ago

Verified

34523d1

<>

Commits on Mar 22, 2020

Merge pull request #1 from ronit-lab/ronit-lab-patch-1

ronit-lab committed yesterday

Verified

dd5d76e

<>

Update .gitignore

ronit-lab / os-316assignment

Watch 0

Star 0

Fork 0

Code

Issues 0

Pull requests 0

Actions

Projects 0

Wiki

Security

Insights

Settings

Branch: master

Commits on Mar 23, 2020

Merge pull request #2 from ronit-lab/ronit-lab-patch-2

ronit-lab committed 41 seconds ago

Verified

74ba6f3

<>

add allocate_map() which will initialize the bit_map character array

ronit-lab committed 1 minute ago

Verified

34523d1

<>

Commits on Mar 22, 2020

Merge pull request #1 from ronit-lab/ronit-lab-patch-1

ronit-lab committed yesterday

Verified

dd5d76e

<>

Update .gitignore

add allocate_map() which will initialize the bit_map character array

master (#2)

ronit-lab committed 5 days ago

Verified

1 parent dd5d76e commit 34523d1f86bb0ce2ce73f31ab87059ec734901

Showing 1 changed file with 5 additions and 0 deletions.

Unified

Split

5 .gitignore

@@ -10,6 +10,11 @@

10

11

12 int bit_map[MAX_PID-MIN_PID]=0;

13

14 int allocate_pid(void){

15 int i,flag=1;

10

11

12 int bit_map[MAX_PID-MIN_PID]=0;

13 + int allocate_map()

14 + {

15 + for(int i=0; i< MAX_PID;i++);



16 + bit_map[i]=0;

17 + }



18

19 int allocate_pid(void){








20 int i,flag=1;

Update .gitignore ronit-lab committed 4 days ago	Verified	 69314b3	<>
switch case to ask user about process ronit-lab committed 4 days ago	Verified	 271f2d3	<>




Commits on Mar 23, 2020

Merge pull request #2 from ronit-lab/ronit-lab-patch-2 ronit-lab committed 5 days ago	Verified	 74ba6f3	<>
add allocate_map() which will initialize the bit_map character array ronit-lab committed 5 days ago	Verified	 34523d1	<>

Commits on Mar 22, 2020

Merge pull request #1 from ronit-lab/ronit-lab-patch-1 ronit-lab committed 6 days ago	Verified	 dd5d76e	<>
Update .gitignore ronit-lab committed 6 days ago	Verified	 73a4a05	<>
Update .gitignore ronit-lab committed 6 days ago	Verified	 b3e49d0	<>
fork() system call can also be used to create process ronit-lab committed 6 days ago	Verified	 3723804	<>
Update README.md ronit-lab committed 6 days ago	Verified	 966043c	<>
Update .gitignore ronit-lab committed 6 days ago	Verified	 ee4010f	<>
Initial commit ronit-lab committed 6 days ago	Verified	 0259c32	<>



os-final-assinment ronit-lab committed 15 hours ago	Verified	 bea3151	<>
allocated memoy sucessfully ronit-lab committed 16 hours ago	Verified	 cc78fa4	<>
removing void creation() ronit-lab committed 20 hours ago	Verified	 e77a525	<>

Commits on Mar 24, 2020

Commits link :-

<https://github.com/ronit-lab/os-316assignment/commits/master>

