|  |  |  |  |
| --- | --- | --- | --- |
| 2022/2 『자료구조』과제 보고서 | | | |
| 제목 | 6장 실습( ) 과제( O ) | 제출일자 | 2022.  10 .    22 . |
| 학번 | 201911608 | 이름 | 김지환 |

|  |
| --- |
| 2. 원형 큐에서 front가 3이고 rear가 5라고 하면 현재 원형 큐에 저장된 요소들의 개수는? (단, MAX\_QUEUE\_SIZE는 8이다.)  (a) 1 (b) 2 (c) 3 (d) 4  -> 원형큐에서 size는 rear - front 이므로 총 2이다.  (b) |
| 4.다음 중 원형 큐에서 공백상태에 해당하는 조건은?  (a) front == 0 && rear == 0  (b) front == (MAX\_QUEUE\_SIZE-1) && rear == (MAX\_QUEUE\_SIZE-1)  (c) front == rear  (d) front == (rear+1) % MAX\_QUEUE\_SIZE  ->  d를 제외한 a, b, c 모두 공백 상태에 해당한다. |
| 6. 다음과 같은 원형 큐에서 (a)에서 (c)까지의 연산을 차례로 수행한다고 하자. 수행이 완료된 후의 큐의 상태를 그려라. 현재 front는 0이고, rear는 2라고 하자.  [ 0 ] [ 1 ] [ 2 ] [ 3 ] [ 4 ]   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | B | C |  |  |   (a) A추가 (b) D추가 (c) 삭제  ->  [ 0 ] [ 1 ] [ 2 ] [ 3 ] [ 4 ]   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | B | C | A |  |   enqueue(&q, A) - front = 0, rear = 3  [ 0 ] [ 1 ] [ 2 ] [ 3 ] [ 4 ]   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  | B | C | A | D |   enqueue(&q, D) - front = 0, rear = 4  [ 0 ] [ 1 ] [ 2 ] [ 3 ] [ 4 ]   |  |  |  |  |  | | --- | --- | --- | --- | --- | |  |  | C | A | D |   dequeue(&q) - front = 1, rear = 4 |
| 8. 원형 큐에 큐에 존재하는 요소의 개수를 반환하는 연산 get\_count를 추가하여 보라. C언어를 이용하여 구현하라.  ->  2번에서 원형 큐의 사이즈는 rear - front 라고 하였다.  int get\_count(QueueType\* q) {  return q->rear - q->front;  }  -> get\_count ADT  #include <stdio.h>  #include <stdlib.h>  #define MAX\_QUEUE\_SIZE 5  typedef int element;  typedef struct {  element data[MAX\_QUEUE\_SIZE];  int front, rear;  } QueueType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init\_queue(QueueType\* q) {  q->front = q->rear = 0;  }  int is\_empty(QueueType\* q) {  return (q->front == q->rear);  }  int is\_full(QueueType\* q) {  return ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front);  }  void queue\_print(QueueType\* q) {  printf("QUEUE(front=%d rear=%d) = ", q->front, q->rear);  if (!is\_empty(q)) {  int i = q->front;  do {  i = (i + 1) % (MAX\_QUEUE\_SIZE);  printf("%d | ", q->data[i]);  if (i == q->rear) break;  } while (i != q->front);  }  printf("\n");  }  void enqueue(QueueType\* q, element item) {  if (is\_full(q)) error("큐가 포화상태입니다");  q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;  q->data[q->rear] = item;  }  element dequeue(QueueType\* q) {  if (is\_empty(q)) error("큐가 공백상태입니다");  q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;  return q->data[q->front];  }  element peek(QueueType\* q) {  if (is\_empty(q)) error("큐가 공백상태입니다");  return q->data[(q->front + 1) % MAX\_QUEUE\_SIZE];  }  int get\_count(QueueType\* q) {  return q->rear - q->front;  }  int main(void)  {  QueueType queue;  init\_queue(&queue);  for (int i = 0;i < 3;i++)  enqueue(&queue, i + 1);  dequeue(&queue);  enqueue(&queue, 10);  queue\_print(&queue);  printf("queue size = %d\n", get\_count(&queue));  return 0;  }  실행결과 |
| 10. 피보나치 수열을 효과적으로 계산하기 위하여 큐를 이용할 수 있다. 만일 피보나치 수열을 순환에 의하여 계산하게 되면 경우에 따라서는 많은 순환 함수의 호출에 의해 비효율적일 수 있다. 이를 개선하기 위하여 큐를 사용하는데 큐에는 처음에는 F(0)와 F(1)의 값이 들어가 있어 다음에 F(2)를 계산할 때 F(0)를 큐에서 제거한다. 그 다음에 계산된 F(b)를 다시 큐에 넣는다. 피보나치 수열은 다음과 같이 정의된다. 큐를 이용하여 피보나치 수열을 계산하는 프로그램을 작성하라.  F(0)=0, F(1)=1  F(n)=F(n-1)+F(n-2)  -> 원형 큐 코드에서 main 함수를 아래와 같이 작성  int main(void) {  QueueType queue;  init\_queue(&queue);  enqueue(&queue, 0); enqueue(&queue, 1);  int n;  printf("몇 번째 피보나치 수를 구하시겠습니까? ");  scanf("%d", &n);  for (int i = 0; i < n; i++)  enqueue(&queue, dequeue(&queue) + peek(&queue));  printf("%d번 째 피보나치 수는 %d 입니다.", n, peek(&queue));  return 0;  }  실행 결과 |
| 12. 태스크 스케줄링 알고리즘으로 A-Steal 알고리즘이 있다. A-Steal 알고리즘에서 각각의 CPU는 자신이 실행할 태스크가 저장된 덱을 가지고 있다. 하나의 CPU가 자신의 태스크를 종료했다면 다른 CPU가 실행할 태스크를 훔쳐서 실행할 수 있다. 이때 다른 CPU의 덱의 끝에 있는 요소를 가져온다. 간단하게 A-Steal 알고리즘을 구현해보자.  #include <stdio.h>  #include <Windows.h>  #define MAX\_QUEUE\_SIZE 10  typedef struct {  int id;  int time;  } element;  typedef struct {  element data[MAX\_QUEUE\_SIZE];  int front, rear;  } QueueType;  void error(char\* message) {  fprintf(stderr, "%s\n", message);  exit(1);  }  void init\_queue(QueueType\* q) {  q->front = q->rear = 0;  }  int is\_empty(QueueType\* q) {  return (q->front == q->rear);  }  int is\_full(QueueType\* q) {  return ((q->rear + 1) % MAX\_QUEUE\_SIZE == q->front);  }  void queue\_print(QueueType\* q, int i) {  printf("CPU %d - (front=%d rear=%d) = ", i, q->front, q->rear);  printf("| ");  if (!is\_empty(q)) {  int i = q->front;  do {  i = (i + 1) % (MAX\_QUEUE\_SIZE);  printf("%d | ", q->data[i]);  if (i == q->rear) break;  } while (i != q->front);  }  printf("\n");  }  void enqueue(QueueType\* q, element item) {  if (is\_full(q))  error("큐가 포화상태입니다");  q->rear = (q->rear + 1) % MAX\_QUEUE\_SIZE;  q->data[q->rear] = item;  }  element dequeue(QueueType\* q) {  if (is\_empty(q))  error("큐가 공백상태입니다");  q->front = (q->front + 1) % MAX\_QUEUE\_SIZE;  return q->data[q->front];  }  element peek(QueueType\* q) {  if (is\_empty(q))  error("큐가 공백상태입니다");  return q->data[(q->front + 1) % MAX\_QUEUE\_SIZE];  }  void start(QueueType\* q, element t, int i) {  enqueue(q, t);  printf("CPU %d - %d번 작업을 추가합니다. 작업시간= %d분\n", i, t.id, t.time);  }  void A\_Steal(QueueType\* q1, QueueType\* q2, QueueType\* q, int i, int j, int k) {  element steal;  if (rand() % 2 == 0) {  if (!is\_empty(q1)) {  steal = dequeue(q1);  enqueue(q, steal);  printf("※※ CPU %d - CPU %d의 %d번 작업을 가져옵니다.\n", i, k, steal.id);  }  }  else {  if (!is\_empty(q2)) {  steal = dequeue(q2);  enqueue(q, steal);  printf("※※ CPU %d - CPU %d의 %d번 작업을 가져옵니다.\n", i, j, steal.id);  }  }  }  void check(int\* t, int time, int i, int\* task, QueueType\* q1, QueueType\* q2, QueueType\* q, int\* b) {  int j = (i == 1) ? 3 : (i == 2) ? 1 : 2;  int k = (i == 1) ? 2 : (i == 2) ? 3 : 1;  if (\*t > 0) {  printf("CPU %d - %d번 작업처리 중 입니다. \n", i, \*task);  --(\*t);  }  else {  if (\*b) {  printf("CPU %d - %d번 작업을 종료합니다.\n", i, \*task);  \*b = 0;  }  if (!is\_empty(q)) {  int prev = \*task;  element nTask = dequeue(q);  \*task = nTask.id;  \*t = nTask.time;  if (!\*b) {  printf("CPU %d - 새로운 작업, %d번 작업을 시작합니다.\n", i, \*task);  \*b = 1;  }  else printf("CPU %d - %d번 작업 종료, %d번 작업을 시작합니다.\n", i, prev, \*task);  }  if(is\_empty(q) && !(\*b)) A\_Steal(q1, q2, q, i, j, k);  }  }  int main(void) {  int minutes = 20;  int total\_tasks = 1;  int task\_time1 = 0, task\_time2 = 0, task\_time3 = 0;  int task1=0, task2=0, task3=3;  int before1 = 0, before2=0, before3=0;  QueueType queue1, queue2, queue3;  init\_queue(&queue1); init\_queue(&queue2); init\_queue(&queue3);  for (int i = 0; i < 10; i++) {  element task;  task.id = total\_tasks++;  task.time = rand() % 3 + 1;  if ((rand() % 3) == 0) start(&queue1, task, 1, task1);  else if ((rand() % 3) == 1) start(&queue2, task, 2, task2);  else if ((rand() % 3) == 2) start(&queue3, task, 3, task3);  }  system("cls");  srand(time(NULL));  for (int clock = 0; clock < minutes; clock++) {  printf("==================================================\n");  printf("현재 시각 = %d\n", clock);  printf("================== 작업 처리 전 ==================\n");  queue\_print(&queue1, 1);  queue\_print(&queue2, 2);  queue\_print(&queue3, 3);  printf("==================================================\n");  if ((rand() % 10) < 3) {  element task;  task.id = total\_tasks++;  task.time = rand() % 3 + 1;  if (rand() % 3 == 0) start(&queue1, task, 1, task1);  else if (rand() % 3 == 1) start(&queue2, task, 2, task2);  else if (rand() % 3 == 2) start(&queue3, task, 3, task3);  }  check(&task\_time1, clock, 1, &task1, &queue2, &queue3, &queue1, &before1);  check(&task\_time2, clock, 2, &task2, &queue3, &queue1, &queue2, &before2);  check(&task\_time3, clock, 3, &task3, &queue1, &queue2, &queue3, &before3);  printf("================== 작업 처리 후 ==================\n");  queue\_print(&queue1, 1);  queue\_print(&queue2, 2);  queue\_print(&queue3, 3);  printf("==================================================\n");  Sleep(3000);  system("cls");  }  printf("전원이 OFF 되었습니다.\n");  while (!is\_empty(&queue1)) dequeue(&queue1);  while (!is\_empty(&queue2)) dequeue(&queue2);  while (!is\_empty(&queue3)) dequeue(&queue3);  return 0;  }  실행결과                    **.**  **.**  **.**    -> 사용 ADT  작업을 추가하는 void start(QueueType\* q, element t, int i)  작업을 가져 오는 A-Steal 알고리즘  void A\_Steal(QueueType\* q1, QueueType\* q2, QueueType\* q, int i, int j, int k)  작업 진행을 하는 void check(int\* t, int time, int i, int\* task, QueueType\* q1, QueueType\* q2, QueueType\* q, int\* b) |