داکیومنت پروژه xv۶

پیاده سازی تشخیص ددلاک

میلاد زارعی ملکی و علیرضا رحیمی

۸ بهمن ۱۴۰۳

پاییز ۴۰۳۱

دانشگاه علم و صنعت ایران

توضیحات اولیه

اول از هر چیزی بایستی در مورد اتفاقی که هنگامی که سیستم عامل بالا میاید حرف بزنیم:

```
resource_struct_buffer = kalloc();

if (!resource_struct_buffer) {
  panic(structs" resource for memory allocate to "Failed);
}

Resource* resources = (Resource*)resource_struct_buffer;

resource_data_buffer = resource_struct_buffer + NRESOURCE * sizeof(Resource);
int each_resource_size = (KSTACKSIZE - NRESOURCE * sizeof(Resource)) / NRESOURCE;

for (int i = 0; i < NRESOURCE; i++) {
  resources[i].resourceid = i;
  resources[i].acquired = 0;
  resources[i].startaddr = resource_data_buffer + i * each_resource_size;
  resources[i].name[0] = 'R';
  resources[i].name[1] = (char)i;
}

graph_node_buffer = kalloc();
if (!graph_node_buffer) {
  panic(nodes" graph for memory allocate to "Failed);
}
```

در قسمت ابتدایی, یک پیج از کرنل به ذخیره سازی اطلاعات گره های گراف اختصاص داده شده است. این اطلاعات شامل اطلاعات مربوط به منابع و اطلاعات مربوط به پردازش ها می باشد. این اطلاعات به صورت زیر ذخیره می شوند:

- اطلاعات منابع: شامل شناسه منبع، آدرس شروع داده های منبع، وضعیت اشغال بودن منبع و نام منبع می باشد.

بعد از این که کل استراکت های مربوطه ساخته شد, مابقی فضا به ریسورس های مربوطه اختصاص داده می شود. هر ریسورس شامل یک آدرس شروع داده های خود می باشد. این آدرس ها به صورت پشت سر هم در فضای مربوطه قرار می گیرند.

شکل ۱: نحوه استفاده از فضای کرنل اختصاص داده شده

همچنین یک فضایی برای ذخیره سازی اطلاعات گره های گراف اختصاص داده شده است. این اطلاعات شامل اطلاعات مربوط به گره های گراف می باشد. این اطلاعات به صورت زیر ذخیره می شوند:

یک پیج برای این مورد اختصاص داده شده که به واسطه یک متغیر ایندکس فعلی ذخیره سازی اطلاعات در بافر نگه داری میشود

درخواست ریسورس

برای قسمت درخواست ریسورس ابتدا بررسی میشود که آیا ریسورس مورد نظر قبلا درخواست شده است یا خیر. اگر درخواست شده باشد پیغام مناسب چاپ میشود و در غیر این صورت چک میشود که آیا ریسورس مورد نظر اشغال شده است یا خیر. اگر اشغال شده باشد یک یال از ریسورس مورد نظر به ریسورس درخواست کننده اضافه میشود و چک میشود که آیا گراف حلقه دار است یا خیر. اگر حلقه دار باشد پیغام مناسب چاپ میشود و یال اضافه شده حذف میشود. در غیر این صورت ریسورس مورد نظر اشغال میشود و یال از ریسورس مورد نظر به ریسورس اشغال شده اضافه میشود.

```
int requestresource(int Resource_ID)
{
  acquired already is resource the if Check //
  resource the to thread current the from edge request a add ,so if //
  graph the in cycle a is there if check and //

  if (is_resource_requested_by(myproc->()tid, Resource_ID)) {
    cprintf( "Thread%d resource requested already %d\n", myproc->()tid, Resource_ID);
    return -1;
  }

  if (is_resource_acquired(Resource_ID)) {
    add_request_edge(myproc->()tid, Resource_ID);
    if (is_cyclic()) {
      cprintf(detected "Deadlock\n");
      remove_request_edge(myproc->()tid, Resource_ID);
      return -1;
    }
  } else {
    Resource* resources = (Resource*)resource_struct_buffer;
    resources[Resource_ID].acquired = 1;
    add_acquired_edge(myproc->()tid, Resource_ID);
  }


return 0;
}
```

آزاد کردن ریسورس

برای قسمت آزاد کردن ریسورس ابتدا بررسی میشود که آیا ریسورس مورد نظر اشغال شده است یا خیر. اگر اشغال شده باشد یک یال از ریسورس مورد نظر به ریسورس درخواست کننده حذف میشود و چک میشود که آیا ریسورس مورد نظر دیگری منتظر آن است یا خیر. اگر منتظر باشد ریسورس آزاد شده و یک یال از ریسورس منتظر به ریسورس اشغال شده اضافه میشود. در غیر این صورت ریسورس آزاد میشود.

```
int releaseresource(int Resource_ID)
{
  this on waiting threads other are there if check Simply //
  thread the to resource the from edge acquired an add and resource the release ,so if //
  resource the release ,not if //
  if (!is_resource_acquired(Resource_ID)) {
    cprintf( "Resource%dacquired not is \n", Resource_ID);
    return -1;
  }

  remove_acquired_edge(myproc->()tid, Resource_ID);

  int next_thread = get_next_thread_waiting(Resource_ID);

  if (next_thread != -1) {
    add_acquired_edge(next_thread, Resource_ID);
  } else {
    Resource* resources = (Resource*)resource_struct_buffer;
    resources[Resource_ID].acquired = 0;
```

```
  }

  return 0;
}
```

چک کردن وجود حلقه

برای چک کردن وجود حلقه از الگوریتم DFS استفاده شده است. ابتدا تمام گره ها را به عنوان گره هایی که بازدید نشده اند در نظر میگیریم. سپس برای هر گره اگر بازدید نشده بود و از آن گره به یک گره دیگر یال وجود داشت و این گره دیگر بازدید نشده بود و یا از آن گره به گره اول یال وجود داشت و این گره دیگر بازدید نشده بود، یک حلقه وجود دارد و در غیر این صورت حلقه وجود ندارد.

```
int is_cyclic_util(int v, int visited,[] int recStack[])
{
  if(visited[v] == 0)
  {
    visited[v] = 1;
    recStack[v] = 1;

    Node* head = Graph.adjList[v];
    while (head) {
      if (!head->vertex) {
        head = head->next;
        continue;
      }
      if (!visited[head->vertex] && is_cyclic_util(head->vertex, visited, recStack)) {
        return 1;
      } else if (recStack[head->vertex]) {
        return 1;
      }
      head = head->next;
    }
  }
  recStack[v] = 0;
  return 0;
}

int is_cyclic()
{
  acquire(&Graph.lock);
  for (int i = 0; i < MAXTHREAD + NRESOURCE; i++) {
    Graph.visited[i] = 0;
    Graph.recStack[i] = 0;
  }
  for (int i = 0; i < MAXTHREAD + NRESOURCE; i++) {
    if (is_cyclic_util(i, Graph.visited, Graph.recStack)) {
      release(&Graph.lock);
      return 1;
    }
  }
  release(&Graph.lock);
  return 0;
}
```

نوشتن و خواندن از ریسورس ها

برای نوشتن و خواندن از ریسورس ها ابتدا بررسی میشود که ریسورس مورد نظر اشغال شده است یا خیر. اگر اشغال شده باشد چک میشود که آیا اندازه داده های مورد نظر از اندازه ریسورس بیشتر است یا خیر. اگر بیشتر باشد پیغام مناسب چاپ میشود و در غیر این صورت داده ها در ریسورس نوشته یا از ریسورس خوانده میشود.

```
 KSTACKSIZE RESOURCE_SIZE #define- ( NRESOURCE* sizeof(Resource)) /NRESOURCE

int writeresource(int Resource_ID,void* buffer,int offset, int size)
{
  resource the the acquired has thread user if check //
  resource the to data the write ,so if //

  if (is_this_thread_owner_of_resource(Resource_ID)) {
    Resource* resources = (Resource*)resource_struct_buffer;
    if (resources[Resource_ID].acquired) {
      if (offset + size > RESOURCE_SIZE) {
        cprintf(overflow "Buffer\n");
        return -1;
      }
      memmove(resources[Resource_ID].startaddr + offset, buffer, size);
      return 0;
    } else {
      cprintf( "Resource%dacquired not is \n", Resource_ID);
      return -1;
    }
  } else {
    cprintf( "Thread%d resource own not does %d\n", myproc->()tid, Resource_ID);
    return -1;
  }

  return 0;
}
int readresource(int Resource_ID,int offset, int size,void* buffer)
{
  resource the the acquired has thread user if check //
  resource the from data the read ,so if //

  if (is_this_thread_owner_of_resource(Resource_ID)) {
    Resource* resources = (Resource*)resource_struct_buffer;
    if (resources[Resource_ID].acquired) {
      if (offset + size > RESOURCE_SIZE) {
        cprintf(overflow "Buffer\n");
        return -1;
      }
      memmove(buffer, resources[Resource_ID].startaddr + offset, size);
      return 0;
    } else {
      cprintf( "Resource%dacquired not is \n", Resource_ID);
      return -1;
    }
  } else {
    cprintf( "Thread%d resource own not does %d\n", myproc->()tid, Resource_ID);
    return -1;
  }
```

```
  return 0;
}
```