Universidad ORT Uruguay
Facultad de Ingeniería

Obligatorio 2 - Sistemas Operativos

Entregado como documentación para obligatorio de Sistemas Operativos

Tadeo Goday - 256686
Federico Rodriguez - 255981
Bruno Silva - 275315

Tutores: Ernesto Silva, Florencia Varela

Noviembre 2022

Declaración de autoría

Nosotros, Tadeo Goday, Federico Rodríguez y Bruno Silva declaramos que el trabajo que se presenta en esta obra es de nuestra propia mano. Podemos asegurar que:

● La obra fue producida en su totalidad mientras construimos el obligatorio de Sistemas Operativos

● Cuando hemos consultado el trabajo publicado por otros, lo hemos atribuido con claridad

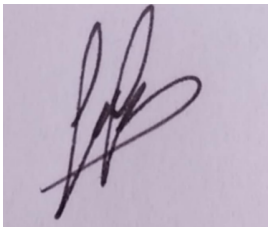
● Cuando hemos citado obras de otros, hemos indicado las fuentes. Con excepción de estas citas, la obra es enteramente nuestra

● En la obra, hemos acusado recibo de las ayudas recibidas

● Cuando la obra se basa en trabajo realizado conjuntamente con otros, hemos explicado claramente qué fue contribuído por otros y qué fue contribuído por nosotros

● Ninguna parte de este trabajo ha sido publicada previamente a su entrega, excepto donde se han realizado las aclaraciones correspondientes

Tadeo Goday
26/11/2022



Federico Rodríguez
26/11/2022

Bruno Silva
26/11/2022

Aclaraciones y decisiones tomadas para las distintas partes del obligatorio

Parte 3: Ada

En la sección main del código sólo está la inicialización de los task BARCO y un generador aleatorio de tiempo para ayudar a que intenten entrar al puerto en cualquier orden. Inicializamos este task 10 veces con un for, y ahí les pasamos por parámetro su número identificador.

Luego, las secciones PUERTO, GRÚA y ATRACADERO DE ESPERA DE ESPERA las implementamos como tasks únicos en loop. Esto nos permite que el acceso a las variables que guardan sea completamente exclusivo para un barco por vez. Es decir, un barco no puede entrar en una sección hasta que el barco anterior termine de entrar o salir.

Al iniciar los barcos, todos deben esperar a que el task puerto les permita entrar para poder luego pedir acceso a la grua, solo los 5 primeros en pedir acceso podrán entrar al puerto y de ahí en adelante el resto espera a que otro barco reduzca el contador del puerto para permitir pasar a otro barco.

Si el puerto les da paso, los barcos piden entonces acceso a la grúa. La grúa tiene un contador que decide cuando entran a la grúa y cuando los manda entonces a esperar al atracadero. Mediante la devolución de un booleano, cada barco sabe si entró en la grúa o si debe esperar a que lo llamen desde el atracadero de espera.

Cuando un barco le toca entrar en la grúa, él espera un tiempo mediante un delay y luego él avisa a la grúa para retirarse y que esta libera a otro barco del atracadero de espera.

Cuando llega un barco a esperar, el task de atracadero de espera controla a qué lugar (ya sea 1 o 2) debe entrar dependiendo de cuál esté libre.

Si los 10 barcos pasan y se retiran del puerto, los task de puerto, grúa y atracadero ya no reciben accept por lo que los loops son terminados acabando con la corrida del programa.

Parte 4: Docker

Creamos en el directorio donde está ubicado el dockerfile una carpeta llamada “app” para guardar dentro el archivo Mostrador.sh. En el dockerfile le indicamos que corra el contenedor con la imagen “alpine:latest” debido a que esta imagen es más que suficiente para correr un script de bash.

Indicamos la carpeta app como el directorio en donde debe trabajar el contenedor y donde se correrán los comandos CMD.

El comando COPY con solo “ . . “ como parámetros hace que copie todo el contenido dentro del directorio donde está el dockerfile hacia el directorio root del contenedor.

Con RUN chmod +x app/Mostrador.sh nos aseguramos de que el script Mostrador.sh tenga el permiso necesario para poder ejecutarse.

Finalmente con el comando CMD [“sh”,”app/Mostrador.sh”] es que se inicializa el script automáticamente con solo comenzar a correr la imagen en el contenedor.

Para construir la imagen, corrimos el siguiente comando:“docker build -t mostrador .” El tag -t, sirve para poder ponerle el nombre que queramos a la imagen.

Tras construir la imagen, para poder interactuar con el script correctamente, debemos agregar al comando run docker los tags “-ti”, de manera que el comando debería quedar de esta manera: “docker run -ti mostrador”.

Pruebas de su funcionamiento: