

Universidad Nacional Autónoma de México
Facultad de Ciencias
Computación Distribuida
Reporte de Práctica 2

1. Equipo

Integrantes del equipo:

- Bonilla Reyes Dafne
- García Ponce José Camilo 319210536
- Ortega Venegas Rodrigo Aldair 318036104
- Rivera Mora Jesús Alberto 313208641

2. Desarrollo

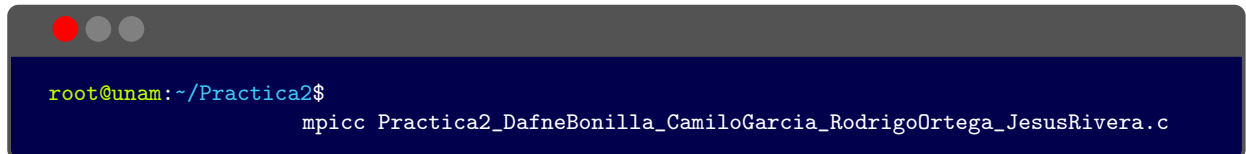
2.1 Descripción del desarrollo de la práctica y su funcionamiento:

- *Configuración:*
 - Se incluyen las bibliotecas necesarias, como `stdio.h`, `stdlib.h`, `mpi.h`, y `time.h`, y se definen algunas constantes con etiquetas: `TAG_IMPOSTOR`, `TAG_PLAN`, `TAG_REY` para identificar los tipos de mensajes enviados.
- *Inicialización:*
 - El programa se inicia con MPI utilizando `MPI_Init` para obtener el número total de procesos en el grupo y el rango de cada proceso (general) en ese grupo.
 - Se verifica si se proporcionan los argumentos correctos al programa. Deben proporcionarse dos argumentos: el número de generales y el número de impostores o traidores. Si los argumentos son incorrectos, el programa muestra mensajes de error y se cierra usando `MPI_Finalize`.
 - Se inicializan las variables `n` e `i` con el número de generales y el número de impostores obtenidos de los argumentos. Se realizan varias comprobaciones para asegurarse de que los valores sean válidos.
- *Generación de mensajes aleatorios:*
 - Se utiliza `srand` para inicializar un generador de números aleatorios con una semilla diferente para cada proceso basada en su rango.
- *Asignación de identificador para cada general.*
 - Se define la variable `leal` para indicar si el general es leal (1) o un impostor (0). El número de impostores se almacena en la variable `sus`.
 - El general 0 decide quiénes son los impostores y se lo comunica a todos los generales. Se utiliza un arreglo impostor para llevar un seguimiento de quiénes son impostores y quiénes no lo son. Los generales impostores se seleccionan de manera aleatoria y se comunican al resto de los generales utilizando `MPI_Send`.
 - Cada general recibe su estado de `impostor` o `leal` del general 0 utilizando `MPI_Recv`.
- *Declaración de variables y arreglo de planes para cada ronda.*
 - Se crea un arreglo `planes` para almacenar los planes (atacar o retirarse) de cada general. Inicialmente, todos los generales eligen un plan de ataque (`planes[i] = 1`) o retirada (`planes[i] = 0`) de forma aleatoria.
 - Se declaran las variables `mayoria = 0`, `cantidad_mayoria = 0`.
 - Se inicializa la variable `necesarios = (size / 2) + sus`; donde `size` es el número de generales y `necesarios` determina como votación válida que el número de votos de la mayoría sea mayor o igual que el valor de esta variable.

- Se declara la variable `rey = 0`, donde se almacena el número del *rank* del rey.
 - Inicializamos la variable ronda con `1 ronda = 1`, esta variable es el contador del ciclo.
 - Inicializamos la variables de finalización del ciclo: `rondas = sus + 1`, el número de rondas es igual al número de generales traidores + 1.
- *Proceso de rondas(Ciclo While).*
- Se inicia un ciclo que se repetirá durante el número de rondas determinado por el valor de `rondas`. En cada ronda:
 - Se elige aleatoriamente un general como rey para esa ronda. Todos los generales actualizan su semilla de generación de números aleatorios para que el rey sea el mismo en todos los procesos. Luego, el general rey envía su plan a todos los demás generales.
 - Cada general envía su plan (o un plan falso si es un impostor) a los demás generales y los recibe.
 - Se calcula la mayoría de planes (ataque o retirada) en función de los planes recibidos. Se determina si la mayoría favorece el ataque o la retirada.
 - El rey comunica su decisión de mayoría al resto de los generales, y los generales la reciben. Si no se alcanza un consenso, los generales usan la decisión del rey como su plan final.
- *Resultados:*
- Después de todas las rondas, cada general determina su plan final y muestra un mensaje indicando si es leal o un impostor, así como su plan final (ataque o retirada).
- *Finalización:*
- El programa MPI finaliza utilizando `MPI.Finalize()`.

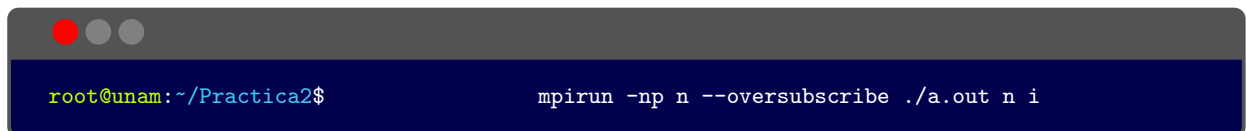
2.2 Compilación y salidas:

- Compilar desde /Practica2:



```
root@unam:~/Practica2$ mpicc Practica2_DafneBonilla_CamiloGarcia_RodrigoOrtega_JesusRivera.c
```

- Para ejecutar desde /Practica2:

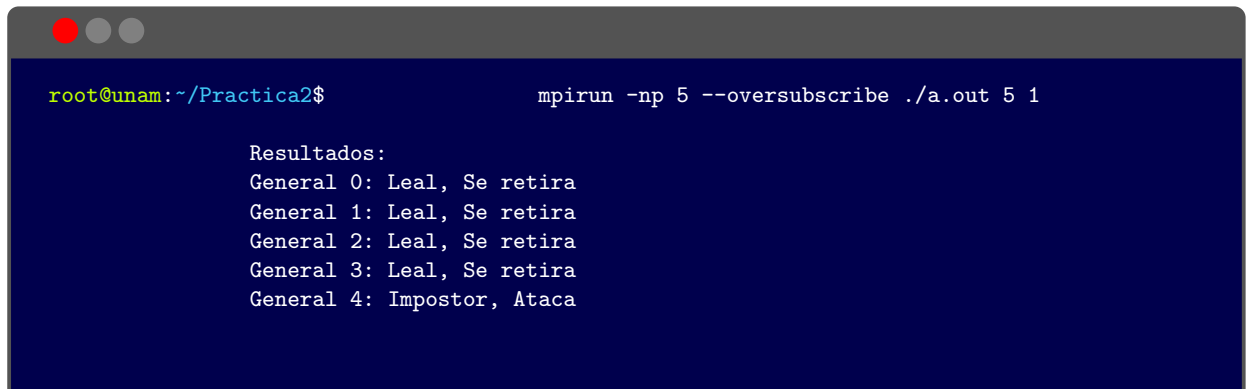


```
root@unam:~/Practica2$ mpirun -np n --oversubscribe ./a.out n i
```

Donde `n` es el número de generales deseados y `i` es el número de impostores deseados.

2.3 Ejemplos de salidas:

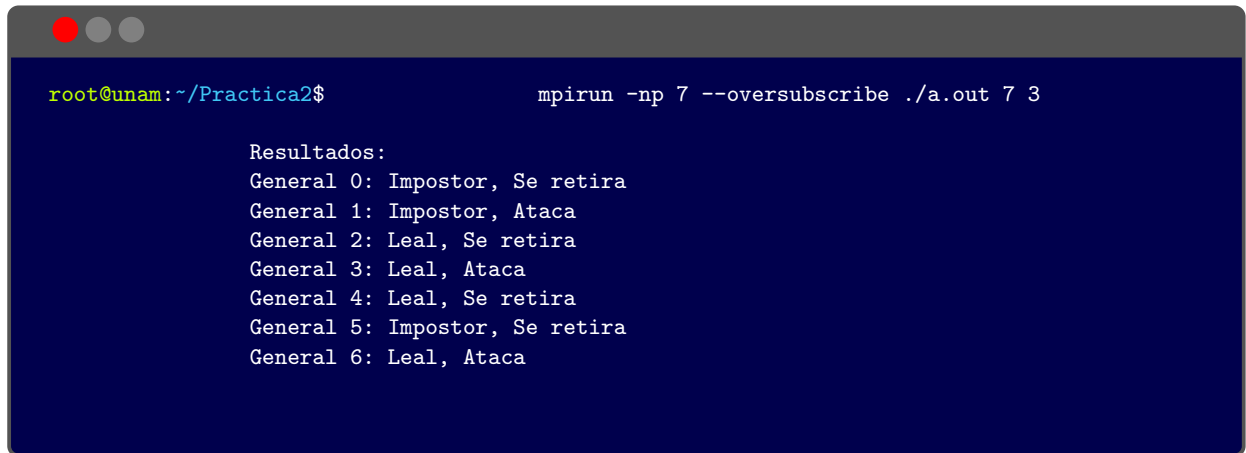
- Ejemplo con 5 generales y 1 impostor:



```
root@unam:~/Practica2$ mpirun -np 5 --oversubscribe ./a.out 5 1

Resultados:
General 0: Leal, Se retira
General 1: Leal, Se retira
General 2: Leal, Se retira
General 3: Leal, Se retira
General 4: Impostor, Ataca
```

- Ejemplo con 7 generales y 3 impostores:

A terminal window with a dark blue background and white text. The window has a title bar with three colored circles (red, yellow, green) on the left. The prompt is 'root@unam:~/Practica2\$'. The command 'mpirun -np 7 --oversubscribe ./a.out 7 3' is entered. The output shows 'Resultados:' followed by seven lines of status for each general.

```
root@unam:~/Practica2$ mpirun -np 7 --oversubscribe ./a.out 7 3

Resultados:
General 0: Impostor, Se retira
General 1: Impostor, Ataca
General 2: Leal, Se retira
General 3: Leal, Ataca
General 4: Leal, Se retira
General 5: Impostor, Se retira
General 6: Leal, Ataca
```

2.4 Comentarios:

Se requiere que el número de generales sea $(4t+1)$, siendo t el número de generales traidores, para que el algoritmo funcione.

Además, es importante mencionar que nos basamos en un pseudocódigo encontrado en internet, por lo tanto en el paso de ver la mayoría del rey, enviamos mensajes para conocer la mayoría del rey y no solo quedarnos con lo que voto (ya que pueden ser diferentes).