Pthread

- **Create** (pthread_create): Crea un nuovo thread.

```
int pthread_create(pthread_t *thread, const pthread_attr_t *attr,
void *(*start_routine)(void *), void *arg);
```

- **Join** (pthread_join): Attende che un thread specificato termini.

```
int pthread_join(pthread_t thread, void **value_ptr);
```

- **Lock e Unlock** (pthread_mutex_lock, pthread_mutex_unlock): Gestiscono l'accesso esclusivo a una sezione di codice tramite mutex.

```
int pthread_mutex_lock(pthread_mutex_t *mutex);
int pthread_mutex_unlock(pthread_mutex_t *mutex);
```

- **Wait** (pthread_cond_wait): Attende che una condizione specificata diventi vera.

```
pthread_cond_wait(&cond, &mutex);
```

- **Barrier**: Coordina i thread, facendoli attendere finché tutti non raggiungono un punto specifico nel codice.

```
int pthread_barrier_init(pthread_barrier_t *barrier, const
pthread_barrierattr_t *attr, unsigned count);
int pthread_barrier_wait(pthread_barrier_t *barrier);
```

- **Condition** (pthread_cond_init, pthread_cond_signal, pthread_cond_broadcast): Variabili di condizione utilizzate per la sincronizzazione tra i thread.

```
int pthread_cond_init(pthread_cond_t *cond, pthread_condattr_t
*cond_attr);
pthread_cond_signal(pthread_cond_t *cond);
pthread_cond_broadcast(pthread_cond_t *cond);
```

- **Mutex** (pthread_mutex_init, pthread_mutex_destroy): Fornisce il locking mutualmente esclusivo per proteggere la sezione critica.

```
int pthread_mutex_init(pthread_mutex_t *mutex, pthread_mutexattr_t
*attr);
int pthread_mutex_destroy(pthread_mutex_t *mutex);
```

- **sleep**: Funzione per sospendere l'esecuzione del thread corrente.

```
void sleep(int seconds);
```

MPI

- **Mpi_Send e Mpi_Recv**: Permettono rispettivamente l'invio e la ricezione di messaggi tra processi.

```
int MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm);
int MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source,
int tag, MPI_Comm comm, MPI_Status *status);
```

- **Mpi_Init e Mpi_Finalize**: Inizializzano e terminano l'ambiente MPI.

```
MPI_Init(&argc, &argv);
MPI_Finalize();
```

- **Mpi_Comm_size e Mpi_Comm_rank**: Restituiscono il numero di processi in un communicator e il rank (identificativo) del processo chiamante all'interno del communicator.

```
MPI_Comm_size(comm, &size);
MPI_Comm_rank(comm, &rank);
```

- **Mpi_Bcast**: Distribuisce un dato dal processo root a tutti gli altri processi nel communicator.

```
MPI_Bcast(void *buf, int count, MPI_Datatype datatype, int root,
MPI_Comm comm);
```

- **Mpi_Barrier**: Sincronizza tutti i processi in un communicator, facendo attendere ciascuno finché tutti non arrivano a questo punto.

```
MPI_Barrier(MPI_Comm comm);
```

- **Mpi_Reduce**: Combina i dati da tutti i processi e li riduce ad un singolo risultato, utilizzando un'operazione specificata (come somma, max, min, ecc.).

```
MPI_Reduce(void *sendbuf, void *recvbuf, int count, MPI_Datatype
datatype, MPI_Op op, int root, MPI_Comm comm);
```

- **Mpi_Scatter e Mpi_Gather**: Distribuiscono (scatter) e raccolgono (gather) dati tra tutti i processi in un communicator.

```
MPI_Scatter(void *sendbuf, int sendcount, MPI_Datatype sendtype, void
*recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm
comm);
MPI_Gather(void *sendbuf, int sendcount, MPI_Datatype sendtype, void
*recvbuf, int recvcount, MPI_Datatype recvtype, int root, MPI_Comm
comm);
```

- **Mpi_Isend e Mpi_Irecv**: Versioni non bloccanti di Mpi_Send e Mpi_Recv, che permettono al processo di continuare l'esecuzione mentre si attende il completamento dell'operazione.

```
int MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest,
int tag, MPI_Comm comm, MPI_Request *request);
int MPI_Irecv(void *buf, int count, MPI_Datatype datatype, int
source, int tag, MPI_Comm comm, MPI_Request *request);
```

- **Mpi_Ssend**: Variante di Mpi_Send che attua un invio sincrono.

```
MPI_Ssend(void *buf, int count, MPI_Datatype datatype, int dest, int
tag, MPI_Comm comm);
```

- **Mpi_Wait**: Utilizzato con operazioni non bloccanti per attendere il loro completamento.

```
int MPI_Wait(MPI_Request *request, MPI_Status *status);
```

- **MPI_Get_count**: Restituisce il numero di elementi ricevuti in una operazione MPI_Recv.

```
MPI_Get_count(MPI_Status *status, MPI_Datatype datatype, int *count);
```

- **MPI_Test**: Testa se un'operazione di comunicazione non bloccante è stata completata.

```
int MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
```

- **MPI_Type_vector**: Crea un nuovo tipo di dati MPI derivato da un tipo esistente.

```
int MPI_Type_vector(int block_count, int block_length, int stride,
MPI_Datatype old_datatype, MPI_Datatype *new_datatype);
```

- **MPI_Type_commit e MPI_Type_free**: Rispettivamente, registra e dealloca un nuovo tipo di dati MPI.

```
int MPI_Type_commit(MPI_Datatype *datatype);
int MPI_Type_free(MPI_Datatype *datatype);
```

- **MPI_Type_create_struct**: Crea un nuovo tipo di dati MPI derivato da un insieme di tipi di dati esistenti.

```
  int MPI_Type_create_struct(int count, int *array_of_blocklengths,
MPI_Aint *array_of_displacements, MPI_Datatype *array_of_types,
MPI_Datatype *newtype);
```

- **MPI_Type_create_resized**: Crea un nuovo tipo di dati MPI derivato da un tipo esistente, con un nuovo intervallo di dislocamento.

```
int MPI_Type_create_resized(MPI_Datatype oldtype, MPI_Aint lb,
MPI_Aint extent, MPI_Datatype *newtype);
```