BÁO CÁO RTOS (FreeRTOS trên STM32)

# 1. Task + Scheduler + Delay

💡 Giải thích:  
- Task: đơn vị công việc độc lập trong RTOS.  
- Scheduler: bộ lập lịch, phân chia CPU dựa trên priority.  
- Delay: cho phép task ngủ một thời gian, CPU rảnh cho task khác.  
  
👉 Đây là nền tảng để có đa nhiệm trong hệ thống nhúng.

📝 Code Demo:

void LedTask(void \*argument) {  
 for(;;) {  
 HAL\_GPIO\_TogglePin(GPIOC, GPIO\_PIN\_13); // Nhấp nháy LED  
 osDelay(500); // nghỉ 500ms  
 }  
}  
  
void UartTask(void \*argument) {  
 for(;;) {  
 char msg[] = "Hello RTOS!\r\n";  
 HAL\_UART\_Transmit(&huart1, (uint8\_t\*)msg, strlen(msg), HAL\_MAX\_DELAY);  
 osDelay(1000);  
 }  
}

# 2. Semaphore

💡 Giải thích:  
- Cơ chế “tín hiệu/chiếc vé” để điều phối lượt.  
- Task chỉ chạy khi giữ được vé.  
- Dùng cho đồng bộ (task ↔ task, task ↔ ISR).

📝 Code Demo:

osSemaphoreId\_t semHandle;  
  
void ProducerTask(void \*argument) {  
 for(;;) {  
 osSemaphoreRelease(semHandle); // phát tín hiệu  
 osDelay(1000);  
 }  
}  
  
void ConsumerTask(void \*argument) {  
 for(;;) {  
 osSemaphoreAcquire(semHandle, osWaitForever); // chờ tín hiệu  
 HAL\_GPIO\_TogglePin(GPIOC, GPIO\_PIN\_13);  
 }  
}

# 3. Mutex (Mutual Exclusion)

💡 Giải thích:  
- Semaphore nhị phân đặc biệt.  
- Bảo vệ tài nguyên chung (UART, biến global).  
- Ai lock thì chính nó mới được unlock.

📝 Code Demo:

osMutexId\_t uartMutex;  
  
void Task1(void \*argument) {  
 for(;;) {  
 osMutexAcquire(uartMutex, osWaitForever);  
 HAL\_UART\_Transmit(&huart1, (uint8\_t\*)"Task1\n", 6, HAL\_MAX\_DELAY);  
 osMutexRelease(uartMutex);  
 osDelay(500);  
 }  
}  
  
void Task2(void \*argument) {  
 for(;;) {  
 osMutexAcquire(uartMutex, osWaitForever);  
 HAL\_UART\_Transmit(&huart1, (uint8\_t\*)"Task2\n", 6, HAL\_MAX\_DELAY);  
 osMutexRelease(uartMutex);  
 osDelay(700);  
 }  
}

# 4. Message Queue

💡 Giải thích:  
- Giống hộp thư để task gửi dữ liệu cho nhau.  
- FIFO, nhiều kiểu dữ liệu (CMSIS v2).  
- Dùng khi task này sản xuất, task kia tiêu thụ.

📝 Code Demo:

osMessageQueueId\_t queueHandle;  
  
void SenderTask(void \*argument) {  
 uint32\_t val = 0;  
 for(;;) {  
 osMessageQueuePut(queueHandle, &val, 0, 0);  
 val++;  
 osDelay(1000);  
 }  
}  
  
void ReceiverTask(void \*argument) {  
 uint32\_t rx;  
 for(;;) {  
 osMessageQueueGet(queueHandle, &rx, NULL, osWaitForever);  
 printf("Received: %lu\r\n", rx);  
 }  
}

# 5. Event Flags

💡 Giải thích:  
- Cho phép task chờ tín hiệu (bit flag).  
- Có thể kết hợp nhiều event cùng lúc.  
- Dùng khi chỉ chạy khi đủ điều kiện.

📝 Code Demo:

osEventFlagsId\_t evtHandle;  
  
void ButtonTask(void \*argument) {  
 for(;;) {  
 if(HAL\_GPIO\_ReadPin(GPIOA, GPIO\_PIN\_0) == GPIO\_PIN\_SET) {  
 osEventFlagsSet(evtHandle, 0x01); // báo hiệu  
 }  
 osDelay(10);  
 }  
}  
  
void LedTask(void \*argument) {  
 for(;;) {  
 osEventFlagsWait(evtHandle, 0x01, osFlagsWaitAny, osWaitForever);  
 HAL\_GPIO\_TogglePin(GPIOC, GPIO\_PIN\_13);  
 }  
}

# 6. RTOS Timer

💡 Giải thích:  
- Hẹn giờ gọi callback định kỳ.  
- Không cần tạo task riêng.  
- Tiết kiệm RAM & CPU.

📝 Code Demo:

void LedCallback(void \*argument) {  
 HAL\_GPIO\_TogglePin(GPIOC, GPIO\_PIN\_13);  
}  
  
osTimerId\_t ledTimer;  
  
void StartDefaultTask(void \*argument) {  
 ledTimer = osTimerNew(LedCallback, osTimerPeriodic, NULL, NULL);  
 osTimerStart(ledTimer, 1000); // 1s toggle LED  
 for(;;) { osDelay(1000); }  
}

# Tổng kết

- Task + Scheduler: trái tim của RTOS, cho phép đa nhiệm.  
- Delay: nhường CPU, tránh block.  
- Semaphore: đồng bộ, chia lượt.  
- Mutex: bảo vệ tài nguyên dùng chung.  
- Queue: truyền dữ liệu giữa task.  
- Event Flags: đồng bộ nhiều điều kiện.  
- Timer: callback định kỳ, tiết kiệm tài nguyên.